FACULTAD DE ESTUDIOS ESTADÍSTICOS

MÁSTER EN MINERÍA DE DATOS E INTELIGENCIA DE NEGOCIOS

Curso 2018/2019

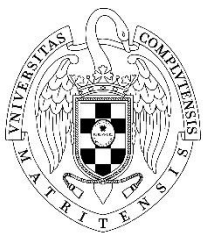
Trabajo de Fin de Máster

Problemas de clasificación de imágenes para la detección de malezas en cultivos de cereales a partir de imágenes aéreas tomadas por drones

***Alumno*: Ana Montano Rodríguez**

***Tutor*: Daniel Gómez González**

Septiembre de 2019



UNIVERSIDAD COMPLUTENSE
MADRID

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

El trabajo que se desarrolla en esta memoria está incoado dentro de lo que se conoce como problemas de agricultura de precisión. El término de agricultura de precisión se refiere a la gestión de parcelas agrícolas sobre la observación, la medida y la actuación frente a la variabilidad inter o intra-cultivo. Es decir, debido a la alta variabilidad que se produce en un mismo campo de cultivo respecto a la producción que se adquiere (ya sea por la forma de aplicar diferentes fertilizantes o herbicidas de manera incorrecta en algunos lugares del perímetro o por una composición distinta del terreno en algunas zonas), poder ser capaces de saber qué es lo que está generando variabilidad dentro de un mismo terreno y encontrar soluciones ante esto. A la hora de aplicar esto, se utilizan diversas tecnologías para encontrar la heterogeneidad de los cultivos, como pueden ser drones o satélites que fotografíen el campo o sensores que tomen diferentes datos en tiempo real sobre características del suelo, por ejemplo, la presencia o ausencia de vegetación o peculiaridades fisicoquímicas. Además, a partir de la agricultura de precisión se pueden obtener otras utilidades como la optimización del cultivo para mejorar la eficiencia de la cosecha, la estimación de la cantidad de fertilizantes a usar, predecir con exactitud el rendimiento y la producción de los cultivos o detectar si un campo contiene o no mala hierba.

En este marco de agricultura de precisión, el trabajo que se desarrolla en este trabajo de fin de máster aborda el problema de detectar malezas en campos de cultivo de cereales. Para cumplir con este objetivo, se utilizarán diferentes métodos de reconocimiento y tratamiento de imágenes tomadas por drones.

¿Para qué podría ser útil este estudio?

Para que al agricultor le sea mucho más fácil saber si la parcela en la que trabaja contiene zonas donde no se producen beneficios por causas de mala hierba y saber en qué puntos debe tomar medidas para obtener cosecha o dejar de cultivar en dichas zonas y así no perder tiempo en plantar semillas donde no serán recibidos los frutos, ni perder dinero en fertilizantes, semillas, etc. Esta es una labor muy eficiente para realizar mediante agricultura de precisión, ya que, un campo de cultivo puede contar con gran cantidad de hectáreas, por lo que resultaría muy lenta y complicada la tarea de que una persona recorra la parcela para detectar si se obtienen zonas de mala hierba y donde se encuentran. Sin embargo, haría mucho más sencilla y veloz esta tarea si se realizase mediante un dron que sobrevolase el perímetro de cultivo y simultáneamente enviara imágenes a un ordenador. Dentro de dicho ordenador, se encontraría alojado un modelo de reconocimiento de imágenes que clasifique si en cada imagen capturada con el dron hay mala hierba o no. Además, también se indica cuáles son las coordenadas exactas donde se encuentran las malezas.

En resumen, con el fin de optimizar las tareas agrarias comentadas en el párrafo anterior, el siguiente estudio se dividirá en dos partes diferenciadas:

1. Clasificar fragmentos de las imágenes provenientes del dron en dos grupos: imágenes con mala hierba e imágenes con cultivo de cereal sano. En esta parte del estudio, se clasifican zonas pequeñas de una imagen grande, esto es denominado sub-imágen.

2. Clasificar todos los pixeles de una imagen para ser capaz de indentificar los puntos especificos de la parcela donde se encuentran las malas hierbas.

Para llevar a cabo ambas partes del estudio, se han utilizado diferentes métodos de tratamiento y reconocimiento de imágenes, desde utilidades básicas de preprocesamiento de imágenes hasta algoritmos complejos de Machine Learning aplicados a la clasificación imágenes.

El procesamiento de imágenes es un campo de la visión artificial que hace que posible que un ordenador sea capaz de obtener información sobre fotografías digitales. Un sistema de imágenes digitales suele distinguirse porque tiene como entrada una imagen y normalmente la salida es un número que da respuesta a la información que se quiere obtener de la imagen en cuestión. Esta información podría consistir en diferenciar los colores que aparecen en la imagen mediante algún modelo de codificación del color, lo que resulta muy útil para detectar las malezas en el cultivo, ya que la mala hierba se diferencia de la producción sana de cereales por el color, entre otras características.

Haciendo referencia a los algoritmos de Machine Learning utilizados para clasificar las zonas de mala hierba y encontrar las coordenadas de estas, serán utilizados métodos estadísticos más sencillos como pueden ser la regresión logística o el algoritmo KNN (K vecinos más cercanos) y algoritmos complejos como redes neuronales simples y Deep Learning o aprendizaje profundo (redes neuronales convolucionales). Todos estos modelos detectarán patrones de las imágenes con cierta aleatoriedad y aprenderán de ellos, consiguiendo clasificar dichos patrones de manera automática.

Cómo es de esperar, para diseñar modelos de aprendizaje profundo, son necesarios gran cantidad de datos que en este estudio no se tienen. Por lo que, un enfoque interesante es comprobar que, si se tienen pocas observaciones, es mejor utilizar modelos más sencillos, aunque se esté trabajando con imágenes.

2. OBJETIVOS

2.1. OBJETIVOS PRINCIPALES

Este proyecto cuenta con un objetivo fundamental que es el de identificar aquellas zonas en las que existe mala hierba. Para llevar a cabo este estudio es necesario abordar dos problemas claramente diferenciados en tratamiento de imágenes: Por un lado encontrar un modelo estadístico que clasifique si una parte concreta de la imagen (sub-imagen) contiene o no mala hierba. Con este modelo se tendrá la información de si en un fragmento de una parcela se encuentra o no mala hierba. Por otro lado, se quiere saber en que coordenadas exactas se encuentra la mala hierba. Para esto último se realizará una clasificación por píxel, es decir, un modelo capaz de clasificar si un píxel contiene o no malezas. Estas dos partes quedan explicadas más detalladamente a continuación:

2.1.1. Parte 1. Modelo de clasificación de sub-imágenes

El problema de clasificar una sub-imagen es similar al problema de clasificación de objetos. Una vez se identifica una zona o sub-imagen debemos clasificarla como zona que contiene mala hierba o zona que no contiene mala hierba, es decir nos encontramos con un problema de clasificación binario. Para abordar este problema, se podrán utilizar todos los algoritmos de clasificación supervisados, pero antes es necesario describir cada sub-imagen en características (variables) que permitan discriminar aquellas imágenes que contienen mala hierba de las que no.

Para lograr este primer objetivo, en la muestra obtenida de imágenes a través del dron, un experto en agricultura ha indicado 1894 zonas (conjuntos de píxeles) en total donde hay o no mala hierba. Por lo que, en un primer lugar se hará un modelo de clasificación de imágenes con trozos de fotos más grandes que están etiquetados. La salida de dicho modelo será si ese conjunto de píxeles o sub-imagen contiene mala hierba o no.

Los algoritmos supervisados utilizados para encontrar el modelo que clasifique a estas sub-imágenes son: Regresión Logística, Redes Neuronales simples y Redes Neuronales Convolucionales. Cabe destacar que estos modelos han sido realizados con los softwares SAS y python

2.1.2. Parte2. Clasificación Pixel by Pixel

Una vez que se ha conseguido identificar una región como target (o lo que es lo mismo, contiene mala hierba) se quiere ser capaz de identificar aquellos píxeles en los que se encuentra la mala hierba, con el fin de identificar, entre otras cosas, si la subzona en la que se encuentra la mala hierba es de gran tamaño o no lo es tanto. Por este motivo, en esta segunda parte del proyecto se va a generar un modelo de clasificación que indique si cada uno de los píxeles de una imagen grande contiene mala hierba o no. Esto se hará entrenando un algoritmo KNN utilizando las sub-imágenes etiquetadas por el experto. Cuando se haya obtenido el modelo que mejor clasifique cada uno de los píxeles, la idea es pasarle una imagen donde aparezca una parcela al completo y devuelva una matriz donde cada elemento corresponda a la probabilidad de que un píxel contenga o no mala hierba. Con esta información se podrá representar un mapa de calor y encontrar las coordenadas donde se encuentren las malezas.

Haciendo un pequeño resumen de los objetivos principales del proyecto, por un lado, se quiere ser capaz de saber si en un campo de cereales hay o no mala hierba y en caso de haberla, saber en qué lugar específico se encuentra.

2.2. OBJETIVOS SECUNDARIOS

1. Comprobar si merece la pena utilizar arquitecturas de aprendizaje profundo para un problema de clasificación binario con pocas imágenes o es preferible hacer modelos estadísticos más sencillos resumiendo cada imagen en una serie de variables que permitan discriminar y clasificar las mismas adecuadamente.
2. El estudio de las diferentes técnicas de visión computacional clásica aplicado a imágenes.
3. El estudio de diferentes métodos de modelización estadísticos aplicados al análisis de imágenes.
4. Ser capaz de describir toda la información que contiene una imagen como un registro convencional para un adecuado tratamiento estadístico de las imágenes (es decir a partir de variables y observaciones).
5. El aprendizaje de diversas librerías de Python como puede ser OpenCV, Keras o Sklearn. Además de tecnologías contemporáneas en Machine Learning como puede ser Google Collaboratory.

3. ESTADO DEL ARTE

3.1. AGRICULTURA DE PRECISIÓN

Como ya se ha comentado en la introducción, el objetivo de la agricultura de precisión es el de controlar todo lo que ocurre en cada una de las parcelas en las que el agricultor está trabajando. Para llevar a cabo esta tarea será necesario disponer de toda la información sobre los campos de cultivo para así poder optimizar al máximo tanto sus cultivos como sus productos de herbicidas y así conseguir una muy alta producción en cada punto.

Hoy en día existen varias formas de llevar a cabo la agricultura de precisión. Una de ellas es la basada en sensores. Esta técnica consiste de la instalación de estos sensores en la maquinaria de agricultura con el fin de que las máquinas realicen operaciones en el campo ajustándose a las necesidades actuales del cultivo. Otra técnica común en la agricultura de precisión es la localización de la maquinaria en el terreno, para ello, se utilizan sistemas GPS y comunicaciones vía satélite para tomar contacto con la maquinaria, el objetivo de es maximizar al máximo las horas productivas ya que se tiene un control preciso de las horas de trabajo de la maquinaria, lo que hace que también se ahorre combustible y se optimice el trabajo. Algo usual también en el marco de la agricultura de precisión es la creación de mapas. Algunos de estos mapas son, por ejemplo, mapas de producción que tratan de representar el rendimiento de cada punto. Para ello debe existir una extracción de datos previa que puede realizarse de varias maneras como mediante sensores, GPS o drones. Otro tipo de mapas que son comunes en esta área son los referentes al suelo, al agua o a las malas hierbas. Esta última metodología explicada en este párrafo puede combinarse con las dos anteriores. Ya que, los sistemas basados en mapas permiten obtener gran cantidad de información sobre

un terreno en comparación a los sensores, la ventaja es que los sensores trabajan en tiempo real, por lo que si se simultanea la labor de amos, se podrá explotar al máximo la producción.

Para llevar a cabo las técnicas explicadas en el párrafo anterior, es imprescindible seguir los siguientes pasos:

1. Recolección de los datos.
2. Procesamiento e interpretación de los datos recolectados.
3. Tomar decisiones en función a las conclusiones obtenidas en el análisis de datos.
4. Aplicación de las medidas necesarias.

La siguiente imagen (figura 1), consiste en un esquema de estos cuatro pasos.

Figura 1.Ciclos de la agricultura de precisión

[1]

Si llevásemos estas cuatro fases al estudio que se desarrolla a continuación, la recolección de datos se efectúa a través de un dron, el procesamiento y la interpretación de la información se lleva cabo a través del análisis de imágenes digitales y en función de las conclusiones obtenidas en el proyecto, la toma de decisiones y la actuación en el campo se dejaría en manos de un experto.

[1] http://www.grap.udl.cat/es/presentacion/ap.html

3.2. LOS DRONES EN AGRICULTURA DE PRECISIÓN

Los vehículos aéreos no tripulados (drones) son una herramienta muy importante dentro de la agricultura de precisión. Un dato interesante es que los japoneses llevan utilizando drones en la agricultura desde hace más de una década para rociar herbicidas y fertilizantes, otro país donde es frecuente el uso de drones para estos fines en Brasil, normalmente para campos de soja. Actualmente se ha aumentado muchísimo su uso debido a que es una herramienta agrícola muy versátil y pueden hacer que se reduzcan significativamente los costes de cultivo además de disminuir el impacto ambiental.

Como ya se ha dicho antes, el cometido que pueden realizar este tipo de dispositivos en cuanto a agricultura se refiere, resulta bastante voluble. Algunos de los datos capaces de recoger en forma de imágenes sobre las condiciones de cultivo se enumeran a continuación.

1. la altura de la planta.
2. el recuento de plantas.
3. la salud de las plantas.
4. presencia de malas hierbas.
5. estimaciones de biomasa relativa.
6. datos 3D.
7. nivel de humedad en el terreno.
8. Supervisión en áreas fumigadas.

Estos drones permiten un control a tiempo real de la cosecha, son capaces de captar las imágenes y la geolocalización del momento del vuelo y transmitir dichos datos de manera inalámbrica a un ordenador, Tablet o teléfono. Este proceso hace posible que se obtenga la información de la parcela al mismo tiempo que el dispositivo sobrevuela el área que se quiere estudiar.

Una vez se tienen las imágenes y los registros sobre la localización de las fotografías, comúnmente, se utiliza el análisis de imágenes.

3.3. PROCESAMIENTO DE IMÁGENES

La visión artificial cumple una misión muy amplia e imprescindible en el mundo actual. Teniendo como objetivo comprender que hay detrás de una imagen digital, el procesamiento de imágenes se ha convertido en un análisis vital en la actualidad. Esto se debe a que dentro de este desarrollo tan acelerado de las tecnologías se encuentran fotografías en todas partes que pueden servir para facilitar tareas que hasta hace poco, las realizaba el humano. Se van a comentar algunas de las labores que se llevan a cabo mediante visión artificial:

- Videovigilancia: Es frecuente utilizar el procesamiento de imágenes para analizar videos de vigilancia, por ejemplo, para saber cuándo se está produciendo un robo en una tienda o también en un comercio para saber cuáles son las estanterías más visitadas por la gente. Otra utilidad del análisis de imágenes en

el área de la videovigilancia en el sector energético es la de detectar fugas en sus instalaciones.

- Sector industrial: En este ámbito se encuentran miles de aplicaciones, algunas de ellas son la detección de objetos en cadenas de montaje, detectar el desgaste de un objeto (por ejemplo, una rueda) por reconocimiento de dimensiones o para cuestiones relacionados con la auditoría de productos en un supermercado.
- Sector salud: Dentro de la sanidad puede resultar muy útil el reconociemiento de imágenes ya que podrían ayudar a detectar anomalías en radiografías mediante modelos de clasificación, por ejemplo, para algún tipo de cáncer.
- Redes sociales: Las redes sociales como Facebook e Instagram contienen integrados sistemas de análisis de imágenes capaces de detectar caras u otro tipo de información dentro de una fotografía.

Además de las aplicaciones enumeradas, existen infinitas que no han sido comentadas y cada vez más, se está empleando el análisis de imágenes para diversas tareas en todo tipo de ámbitos.

Todas estas aplicaciones son realizadas a partir de diferentes técnicas dentro del procesamiento de imágenes, ya que, dependiendo del problema que se quiera resolver se utilizarán unas técnicas u otras. Es decir, no se resolverá de la misma manera un problema de detección de objetos que de clasificación de imágenes, por ejemplo. A continuación, se van a describir algunas de las diferentes metodologías para abordar distintas soluciones a través del procesamiento de imágenes (La información que se presenta sobre las diferentes técnicas de procesamiento de imágenes ha sido obtenida del documento (De la Escalera, 2001)) .

- Realce de las imágenes: Estas técnicas consisten en mejorar el estado de las imágenes, podría servir, por ejemplo, para restaurar imágenes en mal estado.
- Depuración de ruidos: Consiste en la eliminación de elementos no nítidos de la imagen para una mejor interpretación y calidad del estudio de esta. Estos ruidos pueden ser consecuencia de la captura de la imagen o la transmisión de esta. El uso de la eliminación de ruidos suele llevarse a cabo antes de aplicar detectores de bordes.
- Pre-procesamiento de las imágenes: Consiste en, técnicas sencillas de tratamiento de imágenes que se realizan antes de hacer el análisis de lo que hay dentro de la imagen. Algunas de estas técnicas podrían ser, por ejemplo, recortar la imagen, ensancharla, estrecharla, girarla, etc.
- Detección de características: Este ámbito consiste en encontrar ciertos elementos dentro de una imagen. Se puede llegar a detectar desde simples líneas y puntos hasta objetos y formas de mayor complejidad. La manera de conseguir esta extracción de características de una imagen no tiene una solución única, sino, que debe plantearse una manera propia de hacerlo en cada problema.
- Problemas de segmentación: Consiste en dividir una imagen digital en varias partes (grupos de píxeles). La idea es simplificar la representación de una imagen para que sea más fácil de analizar. Se puede utilizar para localizar objetos dentro

de una imagen. Siendo precisos, la segmentación de imágenes consiste en la asignación de un valor a cada uno de los píxeles de la imagen.[2]

- Problemas de clasificación de imágenes: Los problemas de clasificación en procesamiento de imágenes tienen como objetivo conseguir distinguir una imagen con una clase determinada de otra imagen diferente que contenga otra clase diferente. Es decir, conseguir clasificar las imágenes en una categoría diferente según las características que se encuentren dentro de las fotografías. Los problemas de clasificación de imágenes se pueden resolver de manera supervisada o no supervisada. La clasificación supervisada consiste en, extraer patrones de imágenes etiquetadas a priori y clasificar según los patrones y características obtenidos a partir de estas imágenes de las que se tiene información a priori sobre qué clase pertenecen. Por otro lado, la segmentación no supervisada, crea clases clústeres dentro de la imagen.

Existen muchos sistemas más de procesamiento de imágenes que los que se han mencionado, pero, en el caso del proyecto desarrollado, se van a usar métodos de pre-procesamiento, segmentación y clasificación de imágenes.

Algo usual en el procesamiento de imágenes son los problemas de clasificación por píxel, que permiten diferenciar la categoría a la que pertenece cada píxel de la imagen. Esto es útil para encontrar ciertas regiones dentro de una imagen o diversos colores. La manera de hacer una clasificación por píxel consiste en obtener características distintivas de cada uno de los píxeles (variables) y construir un clasificador a partir de una tabla de datos convencional donde las observaciones son cada uno de los píxeles y las variables son los valores de dichas características extraídas, la variable respuesta, será la clase a la que pertenece cada píxel. Por lo que, se tratará de predecir la clase a la que pertenece cada píxel de una imagen a partir de otras variables que se definan en ese píxel. La diferencia que tienen estos métodos con los de clasificación de imágenes, es que, cuando se clasifica una imagen, simplemente se especifica si esa imagen pertenece a una categoría u otra, sin embargo en la clasificación por píxel, se puede diferenciar la región concreta que hace que se clasifique esa imagen en una categoría determinada.

Centrando el marco del procesamiento de imágenes a problemas de agricultura de precisión, se pueden encontrar cantidad de soluciones, como la localización de frutos, detección de áreas desaprovechadas, detección de malezas en el cultivo, detección de hongos, etc. En cuanto a esto, se han encontrado varios ejemplos de estudios relativos al procesamiento de imágenes en agricultura de precisión, algunos de ellos son (Valero Ubierna, 2001), (Miguel, 2015) o (Delgado-Gutiérrez, Herrera-Guillén, Medina-Barragán, & Corredor–Gómez, 2017). En todos ellos se utiliza el procesamiento de imágenes para fines dispares, en uno de los casos para detectar cítricos (utilizando técnicas de detección de objetos), otro para la recolección de fresas, y el último es para identificar líneas de cultivo y malas hierbas en campos de maíz (usando métodos de detección de bordes, líneas, etc.).

El último estudio, con unos objetivos bastante parecidos a los de este, utilizan métodos de visión computacional clásicos a la hora de encontrar las líneas de cultivo y las malas

[2] https://es.wikipedia.org/wiki/Segmentaci%C3%B3n_(procesamiento_de_im%C3%A1genes)

hierbas, es decir, simplemente utilizan técnicas de segmentación de la imagen[3], binarización[4] o detección del color para encontrar los puntos deseados. En este trabajo no se van a utilizar tanto esos métodos, sino que más bien se utilizarán métodos de estadísticos de clasificación de patrones para diferenciar las imágenes y encontrar la posición dentro de la imagen de la hierba en mal estado. Es decir, se utilizarán modelos estadísticos de clasificación de imágenes y de píxeles por separado.

4. METODOLOGÍA Y TECNOLOGÍAS UTILIZADAS

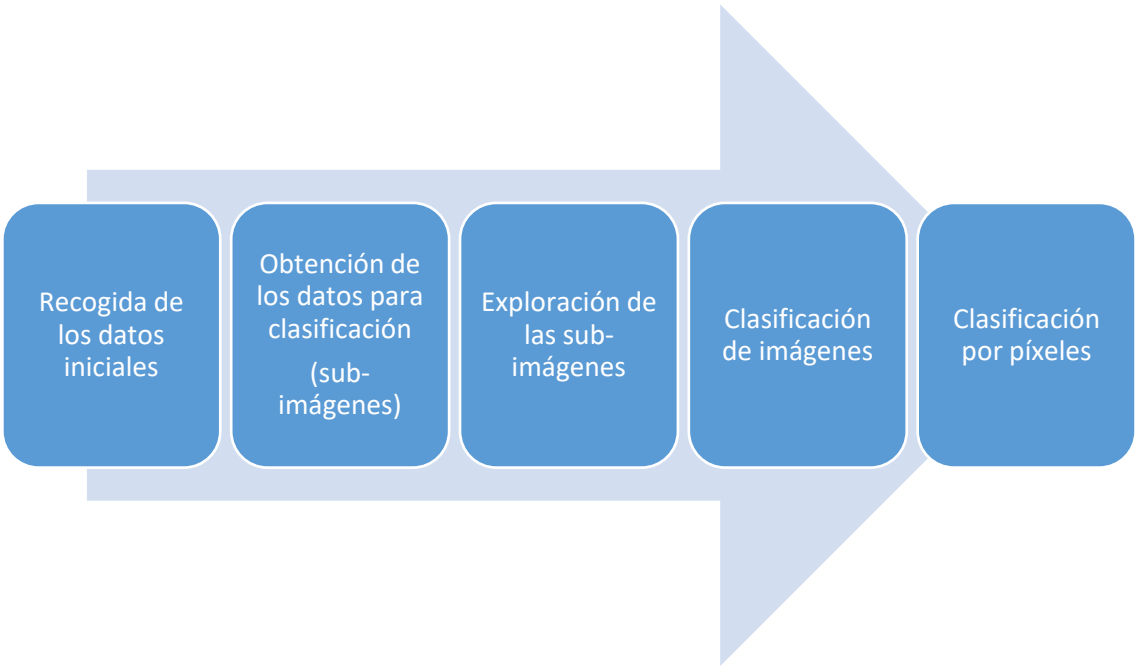
Como se ha comentado anteriormente en los objetivos, la meta del estudio es confeccionar un modelo que sea capaz de distinguir las sub-imágenes (conjuntos de píxeles) con mala hierba, una vez logrado ese modelo, se quiere saber dónde se encuentra esa mala hierba en la fotografía completa de la parcela.

Para llegar a ello, los pasos a seguir durante el análisis, los algoritmos supervisados y los materiales utilizados se detallarán a continuación.

4.1. METODOLOGÍA DE ANÁLISIS

A continuación, se definen los cinco puntos a seguir durante el estudio para cumplir con los objetivos.

Figura 2. Esquema de tareas a llevar a cabo para cumplir con los objetivos



[3] Segmentacion de imágenes: proceso que consiste en dividir una imagen digital en varias partes.
[4] Binarización: Proceso de reducción de la información de una imagen haciendo que solo tomen 2 valores cada uno de los píxeles (true o false).

- Recogida de los datos iniciales: Antes de comenzar a hacer cualquier tipo de análisis se explicará cómo se han obtenido los datos que se tienen en el punto de partida
- Obtención de los datos para clasificar: Obtención de las sub-imágenes para la clasificación de estas. Además, se obtendrán características (variables) que describan cada uno de los rectángulos.
- Descripción y exploración de los datos obtenidos a partir de las sub-imágenes: En este punto se definirán detalladamente los datos que se tienen sobre cada uno de los rectángulos.
- Clasificación de imágenes: Construcción de un modelo de clasificación para las sub-imágenes, se utilizarán y compararán diferentes algoritmos supervisados.
- Clasificación por píxeles: En este punto, se creará un modelo de clasificación de píxeles. Para llegar hasta él se han seguido dos hipótesis diferentes.

4.1.1. RECOGIDA DE LOS DATOS INICIALES

Antes de comenzar a describir los datos que se tienen, es importante tener en cuenta que, una imagen podría ser considerada como un objeto si lo que se quiere es clasificar la imagen. En este caso, se tendrá cada imagen etiquetada según la clase a la que pertenezca como una unidad independiente con respecto al resto de imágenes. Pero también, una imagen podría ser considerada una base de datos con información en cada uno de sus píxeles. Por ejemplo, una imagen grande contiene 120 millones de datos/píxeles cada uno de ellos con información de tres variables RGB. Esta consideración de una imagen sirve, entre otras cosas, para hacer una clasificación por píxel.

En el punto de partida de este trabajo, se cuenta con 28 imágenes grandes (4000x3000 píxeles) de parcelas enteras tomadas en diferentes días por drones. A partir de ellas, se obtienen distintas sub-imágenes dentro de estas fotos donde un experto identifica si existe mala hierba o no. Se debe tener en cuenta que una sub-imagen es un conjunto de píxeles dentro de esa imagen grande. Por lo que, en un principio, se tienen las imágenes y unas coordenadas en forma de píxeles indicando si dentro de esas coordenadas de la parcela hay mala hierba o no.

Es habitual en problemas de agricultura de precisión, tomar imágenes de la misma parcela en días diferentes, esto se debe a la alta variabilidad producida en los datos de este tipo de estudios. Pueden suceder nuevas malezas en poco tiempo o cambiar la forma de las malezas como por ejemplo el color de estas, lo que hace que un estudio realizado con fotos tomadas en un solo día no pudiera servir debido a que los datos estarían muy sesgados.

Para saber que fragmentos de imagen son los que contienen mala hierba, se ha contado con un experto que ha identificado diferentes partes de la imagen que engloba toda la parcela como mala hierba o cultivo de cereal sano. De esta manera, se podrán entrenar los diferentes algoritmos supervisados que se utilizarán para concluir los objetivos, ya que, a través de esta información, se obtienen diferentes imágenes etiquetadas. Desde aqui deberiamos agradecer al equipo de la universidad de Dinamarca, Aarhus University

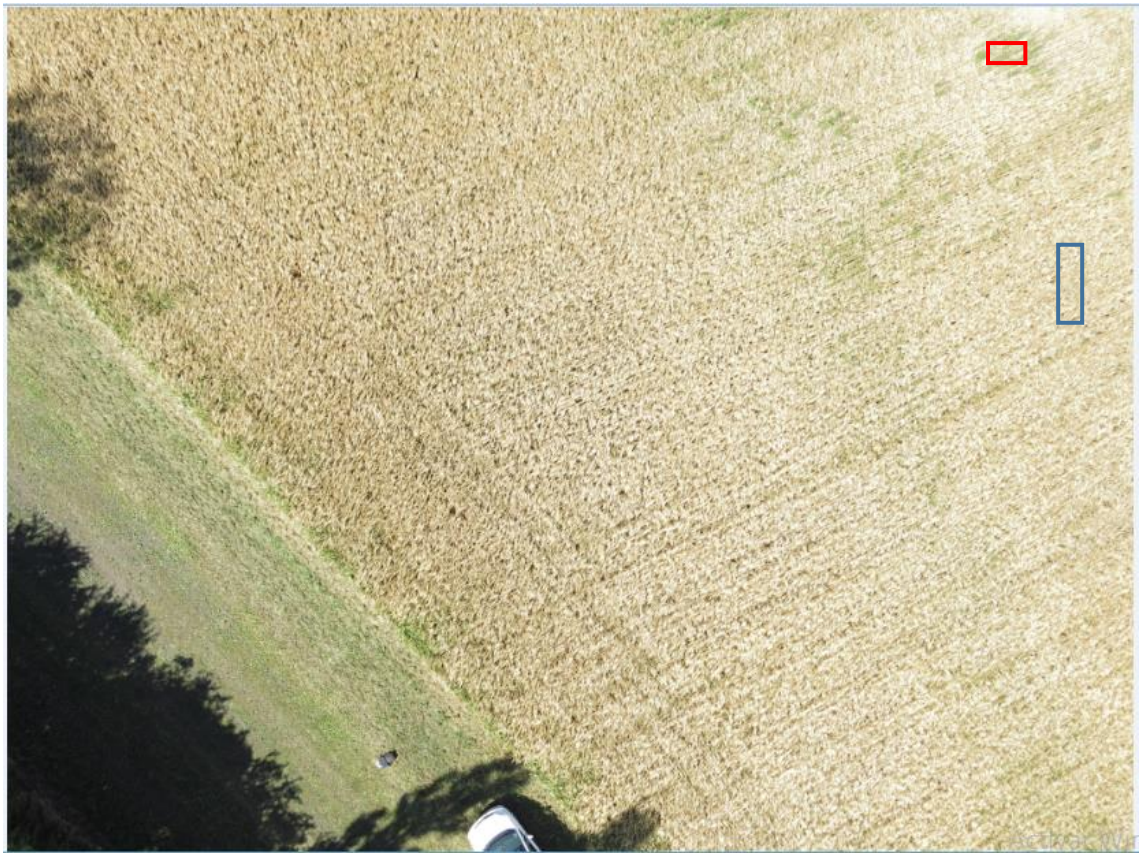
y a Camilo Franco de los Rios por la información suministrada para la realización del trabajo. La información obtenida de cada fragmento clasificado por el experto en el punto de partida del proyecto es la siguiente:

1. El día que fue tomada la imagen completa de la parcela.
2. La imagen a la que pertenece cada uno d ellos fragmentos diferenciados.
3. Las dimensiones e información de todos los pixeles que confeccionan la imagen.
4. La clase a la que pertenece la sub-imagen.

A partir de esta información se deben conseguir cada una de las sub-imágenes como imagen independiente teniendo en cuenta las coordenadas de las zonas catalogadas por el experto.

Para entender mejor la naturaleza de los datos del estudio, la figura 3 es un ejemplo de una de las imágenes completas proporcionadas. Se ha remarcado en rojo, una de las partes donde hay mala hierba, y en azul donde no la hay. Recuérdese que, un fragmento con malezas tiene color verde y, sin embargo, un fragmento con un cultivo sano es de color amarillo.

Figura 3. Ejemplo de imagen



Los rectángulos que se observan en la figura 3, son 2 ejemplos de las sub-imágenes que se obtienen. El rectángulo de color azul estaría clasificado como “No hay mala hierba” o 0 y el rectángulo rojo como “hay mala hierba” o 1.

4.1.2. OBTENCIÓN DE LOS DATOS PARA LA CLASIFICACIÓN (SUB-IMÁGENES)

Una vez que se tiene una idea de los datos iniciales, el siguiente paso será obtener la muestra de los rectángulos, es decir se deben de recortar las imágenes teniendo en cuenta las coordenadas de cada una de ellas.

Para llevar a cabo este proceso, se ha construido un documento en formato Excel con la información de los identificadores de imágenes, los identificadores de los rectángulos, las clases y las coordenadas de estos. Cabe destacar que, para hacer más fácil este cometido, se han organizado las imágenes en una estructura de carpetas determinada para más tarde hacer un bucle que recorra el Excel, entre en las carpetas y recorte las imágenes por los puntos indicados. La tabla 1, es un ejemplo representativo del Excel construido.

Tabla 1. Fichero inicial sobre el conjunto de datos

Carpeta	imagen	Rectangulo	Clase	fila1	col1	fila2	col2	NumFil	NumCol
731	5334	1	0	128	552	180	659	52	107
731	5334	2	0	140	765	288	942	148	177
731	5334	3	0	38	1405	155	1515	117	110
731	5334	4	0	179	2604	239	2753	60	149
731	5334	5	0	368	721	561	873	193	152
731	5334	6	0	385	1063	540	1240	155	177

En la tabla 1, se observan las diferentes variables que se tienen desde el inicio del proyecto, a continuación, se definen cada una de ellas.

- Carpeta: Se trata del nombre de la carpeta donde se encuentran ubicadas cada una de las imágenes.
- Imagen: Consiste en el nombre de las diferentes imágenes.
- Rectángulo: Es el nombre identificador de cada sub-imagen.
- Clase: indica si cada una de las sub-imágenes tiene mala hierba o no. Está codificada de forma binaria donde 1 corresponde a mala hierba y 0 a cuando no hay mala hierba.
- Fila Considerando la imagen como matriz, la variable “fila1” es el número de la fila superior de la coordenada que conforma el rectángulo.
- Col1: Considerando otra vez la imagen como matriz, “col1” es la columna de la sub-imagen, que se localiza más a la izquierda.
- Fila2: Es la fila inferior de la matriz que representa a la imagen.
- Col2: Es la columna colocada más a la derecha dentro de la fotografía.
- NumFil: Es el número de filas que contiene la sub-imagen. Esta variable se obtiene restándole la fila superior a la fila inferior.
- NumCol: Representa el número de columnas del rectángulo. Esta variable consiste en la resta: col2-col1.

Para conseguir los rectángulos por separado y considerarlos como imagen, se utilizarán técnicas de pre-procesado de imágenes como el recorte de la imagen a través de la librería Pillow, para tratamiento de imágenes. La explicación detallada de este proceso se encuentra en el apartado de "Desarrollo del Proyecto".

Una vez se tienen todas las sub-imágenes de forma independiente, se extraerán diferentes características (variables) que describan cada una de las sub-imágenes para así poder realizar análisis posteriores. Entre gran cantidad de variables posibles se ha decidido utilizar el canal de color RGB para analizar las imágenes. Por lo que, de cada una de las sub-imágenes por separado se han obtenido las medias de los canales rojo, verde y azul para así hacer posible la clasificación y diferenciación de la mala hierba. Ya que, una sub-imagen que contenga malezas, se diferenciará en medias de los canales de color de una sub-imagen que que contenga cultivo en buen estado.

4.1.3. DESCRIPCIÓN Y EXPLORACIÓN DE LAS SUB-IMÁGENES

Antes de comenzar cualquier tipo de análisis utilizando los rectángulos obtenidos, es preciso conocer como son las sub-imágenes. La información sobre los rectángulos es útil tanto para la clasificación de imágenes, como para clasificación por píxeles, por lo que es importante saber las características de estos.

En la parte de "Desarrollo del proyecto" se describen detalladamente las variables referentes a las medias de los canales de color Rojo, Verde y Azul de cada uno de los rectángulos para una mejor comprensión de los datos.

4.1.4. CLASIFICACIÓN DE LAS SUB-IMÁGENES

En este punto, como ya se tiene la muestra de sub-imágenes, se puede comenzar con la modelización de los datos para la clasificación de los rectángulos. Antes de ello, se va a hacer la partición de la muestra.

Con el fin de cumplir con el objetivo de encontrar un mejor modelo de clasificación de las sub-imágenes se va a dividir la muestra en entrenamiento, validación y prueba. Debido a que se quieren comparar modelos realizados con diferentes softwares, antes de comenzar el análisis, se han separado de manera aleatoria el 0.03% de las sub-imágenes, esos rectángulos se utilizarán para testear los modelos construidos, así podrán ser comparados los modelos realizados en SAS con los modelos realizados en Python. Por otro lado, los modelos realizados con cada software, se construirán con un 75% de los datos restantes para entrenamiento y un 25 para validación contando siempre con la misma semilla aleatoria dentro de cada software. Así, se compararán los modelos de un mismo tipo entre ellos y los ganadores de cada clase se compararán con la muestra separada al principio.

En cuanto a la elaboración de los modelos de clasificación, los algoritmos utilizados son: Regresión Logística, Redes Neuronales Simples, Redes Neuronales Complejas y Redes Neuronales Convolucionales (el funcionamiento y definición de todos estos algoritmos se explican en apartados de a continuación). Dependiendo de la complejidad de los algoritmos, se ha utilizado un software u otro, la Regresión logística y las redes neuronales simples se han realizado con SAS y las Redes Neuronales Complejas y Convolucionales se han confeccionado con el software libre Python.

Cabe destacar que, en los modelos más sencillos, el input de los datos a través de los cuales se construye el clasificador consiste en las variables que describen las características de la sub-imagen, es decir, se detectarán los diferentes patrones que diferencian una clase de otra a través de la media de los canales RGB para cada rectángulo en formato tabla convencional. Sin embargo, a la hora de construir los modelos complejos, el input de los datos provendrá de la propia sub-imagen, convirtiendo los rectángulos en matriz de píxeles y colocándolos en un formato determinado para el funcionamiento de modelado de análisis de imágenes de la librería "keras". Esto implica que el input de los datos para los modelos más complejos pesará mucho más ya que se tienen tantas observaciones como píxeles haya en el total de sub-imágenes en un formato no tan habitual como el de la tabla de datos convencional. Este es uno de los motivos por los que no se ha utilizado validación cruzada a la hora de comparar los modelos más complejos pero si en los modelos más simples, ya que el tiempo de ejecución se hubiera elevado considerablemente.

4.1.5. CLASIFICACIÓN PÍXEL BY PÍXEL

Una vez construido el modelo clasificador de rectángulos, otro de los objetivos primarios era encontrar las malezas dentro de la imagen completa. Para ello se utilizará otra técnica estadística para crear un modelo que clasifique cada uno de los píxeles como mala hierba o no. De esta manera, se podrá obtener información sobre la localización de las malezas.

El algoritmo utilizado para la elaboración del clasificador de píxeles es el KNN (K- nearest neighbors o en castellanos, k-vecinos más cercanos), cuyo funcionamiento se detalla en el punto 4.1.10.

Es importante señalar que a diferencia de la información que se contaba con los rectangulos donde el experto clasificaba cada uno de ellos como 0 o 1 (no contiene mala hierba o si la contiene), no disponemos de una muestra de entrenamiento para el conjunto de píxeles. Los patrones de colores que deben seguir los pixeles con mala hierba o los que no lo son, han sido identificados automaticamente en base a las subimágenes. Por este motivo, no es posible validar la bondad del modelo de clasificación pixel by pixel al no disponer de un "ground truth" real sino artificial. La validación de esta última fase de clasificación se ha realizado visualmente con los expertos.

A la hora de componer este modelo, se siguen dos hipótesis en cuanto a los píxeles que constituyen cada uno de los rectángulos, debido a que no se tiene información de la clase a la que pertenecen los píxeles en particular.

La primera hipótesis, afirma que los píxeles contenidos en una misma sub-imagen no tienen por qué ser todos de la misma clase que la sub-imagen, es decir, aunque la mayoría de los píxeles contenidos en una sub-imagen con mala hierba sean de la clase "mala hierba" puede haber algunos que pertenezcan a la otra clase restante. En este caso, no se pueden usar los mismos píxeles de la sub-imagen para entrenar el KNN. Para suavizar este problema y asumiendo cierta mayoría de pixeles mala hierba (ya que los rectángulos escogidos no eran de gran tamaño) se ha utilizado el valor de la media de los canales RGB de la sub-imagen como posible patrón. La metodología que seguir en este caso sería entrenar el modelo KNN con las medias de los canales RGB y una vez obtenido el modelo, pasarle una imagen completa en forma de matriz de píxeles con los valores de los canales RGB en cada elemento y que devuelva otra matriz con las probabilidades de que cada uno de los píxeles contengan malezas o no, después con esa matriz, se representará un mapa de calor.

La segunda hipótesis consiste en asumir que todos los píxeles de una sub-imagen son de la misma clase que la sub-imagen, lo que quiere decir que, si un rectángulo pertenece a la clase "mala hierba" querrá decir que todos los píxeles que forman ese rectángulo también pertenecerán a esa clase. La manera de elaborar el clasificador de píxeles en este caso consiste en entrenar el KNN mediante los píxeles de los rectángulos clasificados en función a la clase a la que pertenecen dichos rectángulos, teniendo para cada uno de esos píxeles el valor que obtienen los canales RGB. En este caso, también se representará un mapa de calor de las probabilidades de que cada uno de los píxeles de la imagen completa sea de la clase "mala hierba" o "no mala hierba".

En ambos casos, no es posible evaluar los modelos con las clasificaciones predichas por píxeles, ya que no se tiene ninguna imagen con la información correspondiente a la clase a la que pertenecen cada uno de sus píxeles. Pero si se puede evaluar lo bien que clasifica el modelo a las medias de los canales RGB según la clase bajo la primera hipótesis y bajo la segunda hipótesis, lo bien que clasifica cada uno de los píxeles contenidos en una sub-imagen teniendo en cuenta que todos los píxeles del rectángulo pertenecen a esa clase. Para ello, en ambos casos se dividirán cada una de las muestras en un 80% para entrenamiento y un 20% para validación.

4.1.6. TRATAMIENTO DE IMÁGENES

"Una imagen digital es una representación bidimensional de una imagen a partir de una matriz numérica."[5] Una imagen digital puede ser fácilmente modificable a partir de filtros para eliminar elementos de la imagen, remarcarlos, cambiar el tamaño de la imagen, etc.

[5] https://es.wikipedia.org/wiki/Imagen_digital

Cuando se habla de procesar imágenes a color, quiere decir que a cada píxel le corresponden tres valores de intensidad (RGB) en lugar de uno que correspondería si la imagen fuese en blanco y negro.

Los tres canales RGB (Red, Green, Blue) constituyen de un modelo de color mediante el cual es posible representar colores con la mezcla de estos.

Figura 4.Canal de color RGB

La figura 4, es una representación del modelo de color RGB. Los colores que no son el rojo, el verde o el azul, se forman a partir de la suma de estos.

Por tanto, cualquier imagen digital de dimensiones nxm puede ser modelizada como tres matrices nxm: IR, IG y IB donde el elemento (i,j) asociado a cada una de esas matrices representa la intensidad del pixel con coordenada (i,j) en cada uno de los colores. A partir de estos valores, se puede obtener muchísima información sobre la imagen y entrenar modelos capaces de clasificar imágenes o conjuntos de píxeles, además de predecir nuevos píxeles o detectar objetos dentro de la imagen.

Existen otros modelos de colores más complejos que el modelo de color RGB. Pero estos no serán utilizados en el siguiente estudio.

Dentro del procesamiento de imágenes también existen otros métodos que modifican el estado de la imagen como puede ser el redimensionado, que consiste en ensanchar o estrechar la imagen cambiando su número de píxeles (si se aumenta el número de píxeles, esta se ensancha y si se disminuye se estrechará). Otros métodos útiles de procesado de imagen serían el recorte de la imagen o la rotación de esta.

Para cumplir los objetivos del trabajo, solo se va a utilizar el modelo de color RGB y algunos métodos de procesado de imágenes de redimensionado de la imagen o los recortes.

4.1.7. MACHINE LEARNING

El Machine Learning es un método de análisis de datos que busca la automatización de modelos analíticos o modelos de análisis de datos.en sistemas que aprenden

automáticamente, es decir, identificar patrones complejos a través de los datos, utilizando diferentes algoritmos, que, con el tiempo, son capaces de hacer predicciones a futuro.

Para que un algoritmo sea capaz de aprender de un conjunto de datos, es común dividir la muestra en dos partes, entrenamiento y validación. La muestra de entrenamiento será utilizada para crear los modelos que, en el caso del siguiente estudio clasificarán los fragmentos con mala hierba y los que no. El conjunto de datos de validación será el utilizado para evaluar la bondad de ajuste de dicho modelo. Es decir, con datos que no se han utilizado para construir los modelos, se va a comprobar como de bueno es, estudiando diferentes estadísticos como puede ser el "*Área Bajo la Curva de ROC*" o algunas tasas de fallos o aciertos (estos índices se explicarán más a fondo más tarde). Por otro lado, para comparar los diferentes modelos entre sí, se utilizará la técnica "Validación Cruzada". Esta técnica consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones, de esta manera, queda demostrado que los resultados del análisis no dependen de la partición que se haya realizado.

Los algoritmos a utilizar para cumplir con los objetivos del proyecto son: Regresión logística, Redes Neuronales simples, redes neuronales profundas y KNN (k-nearest neighbours o k-vecinos más próximos). Estos algoritmos se van a explicar uno a uno en los siguientes apartados.

4.1.8. REGRESIÓN LOGÍSTICA

El análisis de regresión logística es un tipo de regresión encuadrada en los modelos lineales generalizados (GLM). Esta consiste en explicar una variable respuesta binaria a partir de variables independientes que pueden ser continuas o discretas.

A continuación se muestra la fórmula del modelo logit, una parte de la regresión logística que será utilizada en el estudio. Se trata de un modelo de clasificación muy sencillo que muestra la probabilidad de que ocurra un evento.

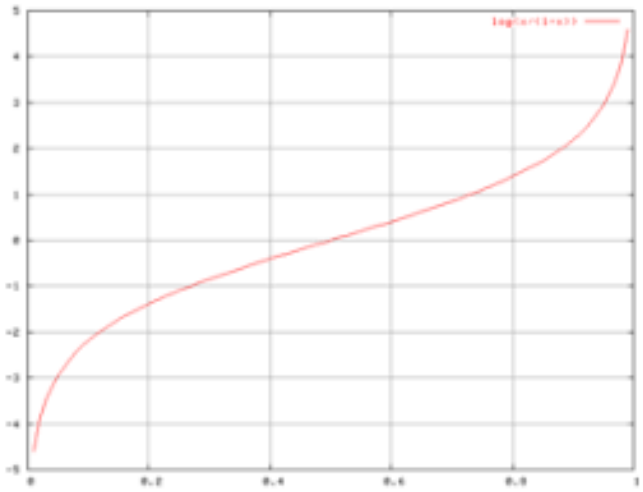
$$\operatorname{logit}(p_i) = \ln\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_{1,i} + \cdots + \beta_k x_{k,i}.$$

$$p_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_{1,i} + \cdots + \beta_k x_{k,i})}}$$

En la fórmula anterior, se ve que el logit de un número p_i consta de una función de probabilidad que se define como la expresión mostrada anteriormente.

La función logística cumple las propiedades de ser monótona creciente, acotada en un intervalo [0,1] y su representación gráfica es la siguiente:

Figura 5.Función Regresión Logística



Su gráfica es de una onda sinusoidal donde, el valor mínimo de la variable dependiente tomaría el valor 0 y el máximo valor que podría tomar sería 1.

En este caso, los modelos de regresión logística serán utilizados para encontrar un mejor modelo clasificador de las sub-imágenes utilizando como variables independientes la media de r de g y de b.

4.1.9. REDES NEURONALES

Las redes neuronales son un tipo de modelo matemático que imita al comportamiento de un cerebro humano. Estos modelos son capaces de resolver problemas complicados de predicción y de clasificación.

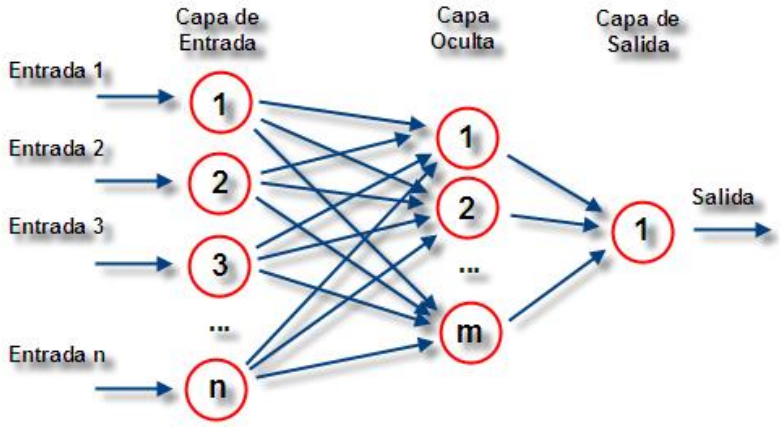
En cuanto a la estructura de estas, existen unas unidades denominadas nodos o elementos procesadores que se asemejan a una neurona del cerebro, que, realizan diversas funciones como puede ser: La evaluación de las señales de entrada, la suma de las mismas o la comparación con el umbral de salida. Normalmente, las neuronas o nodos se agrupan en unidades llamadas capas y suelen ser del mismo tipo dentro de dichas capas. Pueden distinguirse 3 tipos de capas.

- **Capas de entrada:** Estas son las que reciben los datos a predecir o clasificar.
- **Capas de salida:** Proporcionan la respuesta de la red en función de los datos de entrada.
- **Capas ocultas:** Se encargan del procesamiento de la propia red de forma interna.

Por otro lado, existen conexiones entre neuronas. Estas conexiones suelen producirse entre neuronas de distintas capas, es decir, las neuronas de la capa de entrada se relacionan con las neuronas de las capas ocultas y estas a su vez con las neuronas de la capa de salida.

La siguiente figura es un esquema de una red neuronal sencilla con una capa de entrada, una de salida y una capa oculta.

Figura 6. Estructura de una Red Neuronal



[6]

El concepto **pesos**, indica la fuerza que tiene la conexión entre dos nodos. Por esto mismo, algunas entradas en un mismo nodo pueden tener mayor magnitud que otras.

Si se habla de aprendizaje de una red neuronal, simplemente se hace referencia al ajuste de los parámetros, es decir, determinar un conjunto de pesos que permita a la red realizar el procesamiento deseado. Existen varios tipos de aprendizaje para una red neuronal, pero solo será utilizado el aprendizaje supervisado en este caso.

Los algoritmos de optimización son muy importantes en la fase de aprendizaje, ya que gracias a estos la red aprende. Estos utilizan diferentes métodos para que la red se ajuste a los datos para obtener los mejores resultados posibles.

Debido a que la salida de los diferentes nodos de la red puede ser un valor o un conjunto de valores no comprensible, existen las funciones de activación que transforman dichos valores en un rango de salida determinado como (0,1) o (-1,1) normalmente.

En la fase 1 de clasificación de este trabajo (clasificación de las imágenes), se han realizado redes neuronales simples, jugando con el número de nodos (variando diferentes números para ver como de bien clasifica la red), las funciones de activación (probando todas aquellas que ofrece el software utilizado), y los algoritmos de optimización (variando también todos los que ofrece el software).

A continuación, son explicadas las distintas funciones de activación y algoritmos de optimización utilizados.

- **Funciones de activación:**

- Tangente Hiperbólica (tanh): Esta función de activación transforma los valores introducidos en un rango acotado entre -1 y 1, donde los valores más altos tomarán valores cercanos a 1 y los más bajos a -1.
- Softmax: Transforma las salidas a probabilidades.

[6] https://esacademic.com/dic.nsf/eswiki/994505

- Logarítmica sigmoide: Devuelve valores entre 0 y 1
- Arcotangente: Transforma todos los valores de entrada en valores acotados en el intervalo real ($-\frac{\pi}{2}$, $\frac{\pi}{2}$)
- Seno: Transforma todos los valores introducidos en valores comprendidos entre el rango de -1 y 1.
- Logística: También muestra valores en forma de probabilidad ente 0 y 1.

- **Algoritmos de optimización:**

- Back Propagation: Se trata de uno de los más sencillos. Es un método del cálculo del gradiente que, emplea un ciclo de propagación en 2 fases, por un lado, la señal de entrada se propaga por toda la red, hasta que, la seña de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas. En la segunda fase, las señales de error obtenidas se propagan hacia atrás, capa por capa hasta llegar a la capa de entrada pasando por las capas ocultas. Esto permite que las neuronas aprendan a reconocer errores y sean capaces de eliminar el ruido para obtener los resultados.

- Levenberg-Marquardt (levmar): Es un algoritmo basado en mínimos cuadrados no lineales.

- Quasi-Newton (Quanew): Son métodos que se utilizan para encontrar ceros o máximos y mínimos locales.
- Trust Region Method (trureg): Es uno de los algoritmos de optimización más comunes en problemas no lineales. Consiste en definir una región alrededor de la mejor solución actual donde un modelo se aproxima en cierta medida a la función objetivo original.

4.1.10. DEEP LEARNING Y REDES NEURONALES CONVOLUCIONALES.

El Deep Learning o aprendizaje profundo, es un campo de la inteligencia artificial que utiliza redes neuronales convolucionales para obtener resultados de predicción o clasificación llevando a cabo un proceso de aprendizaje que imita al del cerebro de los seres humanos.

Para el caso del reconocimiento de imágenes en Deep Learning se utilizan las redes neuronales convolucionales:

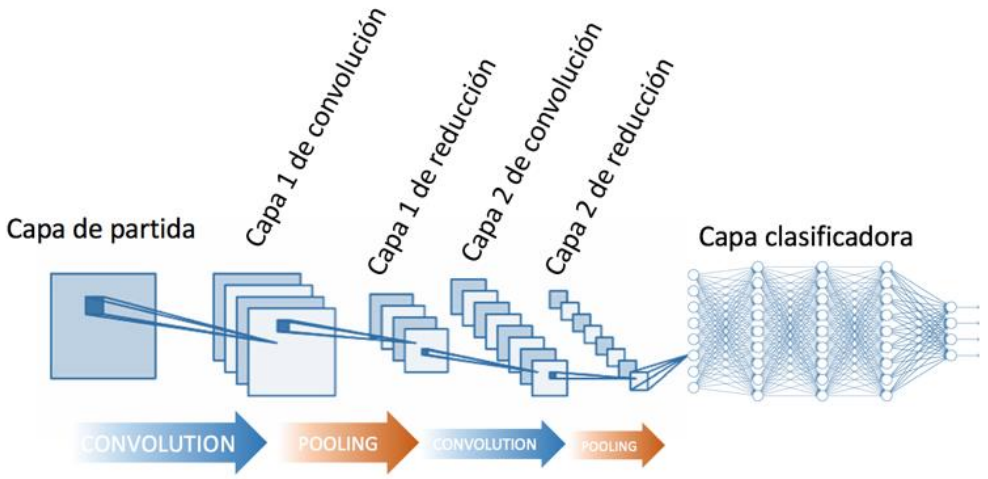
“Una red neuronal convolucional es un tipo de red neuronal artificial donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (V1) de un cerebro biológico. Este tipo de red es una variación de un perceptrón multicapa, sin embargo, debido a que su aplicación es

realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones."[7]

Como bien dice el párrafo anterior, estas redes son muy potentes para el análisis de imágenes ya que son capaces de detectar características simples en el paso de las imágenes por sus capas, como son: la detección de bordes, líneas, etc. y poco a poco encontrar estructuras más complejas hasta encontrar lo que se está buscando. Para ello, la red neuronal convolucional contiene varias capas ocultas especializadas y con cierta jerarquía, es decir, las primeras capas detectarían las líneas, bordes o curvas y van especializándose hasta que las últimas capas más profundas detectarían estructuras muy complejas como son caras u objetos.

Debido a que la red neuronal convolucional extrae caracteristicas de la imagen de manera automatica, y todas las imágenes han de pasar por todas sus capas, para entrenar este tipo de redes se necesita una gran cantidad de imágenes. La estructura de una red neuronal convolucional es la siguiente:

Figura 7. Estructura de una Red Neuronal Convolucional



[8]

La figura 7 muestra gráficamente la estructura de una red neuronal convolucional. Como se puede ver, estas se componen de diferentes capas denominadas convolucionales y de reducción además de la capa de partida y la capa clasificadora. El funcionamiento de estas y el proceso que se realiza se describe a continuación:

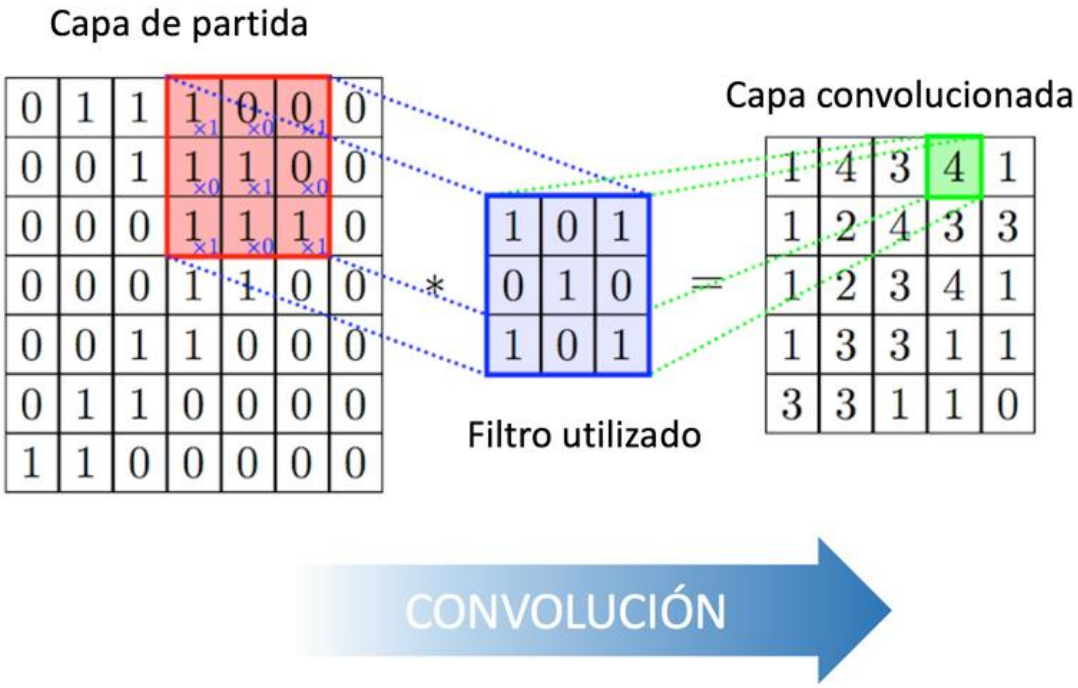
La **capa de partida** tomará como entrada el valor de los píxeles, que previamente deben ser normalizados. Los píxeles toman valores de 0 a 255 (valores que toman en función del canal RGB), por lo que antes de pasar las imágenes por la red se estandarizarán dividiendo los píxeles por 255 haciendo que definitivamente estos tomen valores de 0 a

[7] https://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/
[8] http://www.diegocalvo.es/red-neuronal-convolucional/

1. En resumen, la capa de entrada tomará una matriz de píxeles estandarizados con la información de los tres canales de color Red, Green, Blue.

En las **capas convolucionales** se "agrupan" los píxeles cercanos de las capas de entrada y se realiza el producto escalar contra una pequeña matriz que se llama kernel (en el caso de que las imágenes estén a color se tendrá un filtro con 3 kernels, posteriormente esos kernels se suman obteniendo una sola salida). Dicha matriz kernel toma valores aleatorios y poco a poco se van ajustando mediante backpropagation. Ese kernel recorre todas las matrices de input de arriba abajo y de izquierda a derecha obteniendo una nueva matriz de salida. Cabe destacar que no se tiene un solo kernel, sino muchos kernels, el conjunto de estos kernels se denomina filtro. Al final se obtendrán tantas matrices como filtros haya y a estas matrices resultantes se les denomina mapa de características o en inglés "feature mapping". La figura 8, es un ejemplo visual de cómo se obtiene ese mapa de características.

Figura 8.Capa de convolución



[9]

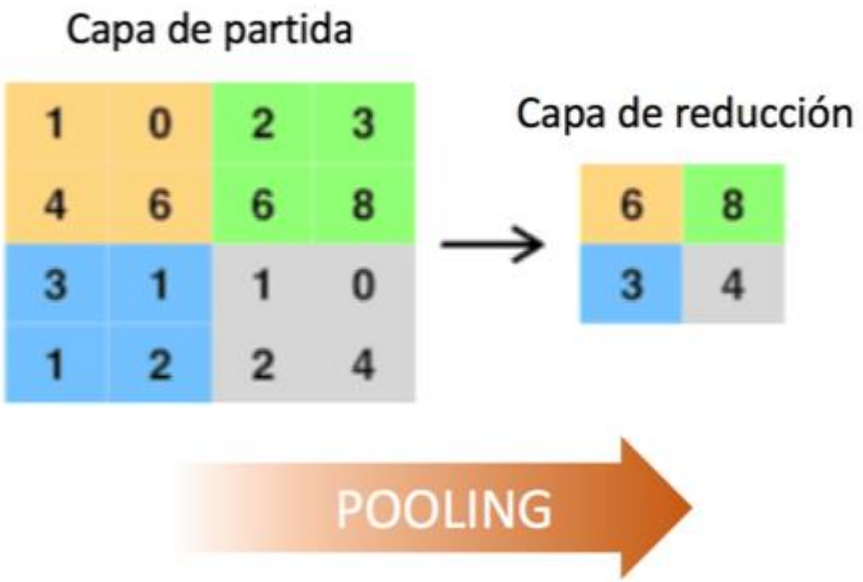
Una vez obtenidos los mapas de características se le aplican la función de activación, que en estos casos se suele utilizar la función de activación ReLu (Rectifier Linear Unit) que es de f(x) = max(0,x). Esta función de activación transforma los feature mapping para que sirvan de input en las siguientes capas.

Una vez obtenido el output de la capa convolucional se entra en la capa "puling" o **capa de reducción**. En este tipo de capas se disminuyen la cantidad de parámetros realizando otra matriz que resume las características más importantes. Normalmente la manera de reducir estos parámetros es mediante la técnica llamada "Max-Pooling", esta consiste

[9] http://www.diegocalvo.es/red-neuronal-convolucional/

en, por ejemplo, si quisiéramos reducir nuestro mapa de características a una matriz 2x2, se recorre la matriz en vez de píxel por píxel en 2x2 píxeles y se va recogiendo el mayor valor de esos 4 píxeles (en caso de que la matriz output deseada sea 2x2). Al final se obtendrá una matriz de 2x2 en este caso por cada imagen introducida en la red. Esto hace que se simplifique bastante la información y se sigue recogiendo lo más importante de cada una de las imágenes. La figura 9 consiste en un ejemplo de Max-Pooling.

Figura 9. Capa de reducción



[10]

Este proceso se realiza tantas veces como capas convolucionales y de pooling contenga la red. Finalmente una red neuronal convolucional contiene la capa de clasificación. Esta es una red neuronal simple a la que se le suele aplicar la función de activación softMax y cuya última capa tendrá la cantidad de neuronas equivalente al número de clases de la clasificación (en el caso de los campos con mala hierba serían 2 neuronas). En esta última red simple se obtendrá el valor de la predicción, en este estudio, se obtendrá 1 si la imagen tiene mala hierba y 0 en caso contrario. La función de activación SoftMax pasa a probabilidad entre 0 y 1 las predicciones obtenidas.

Existen diferentes estructuras de redes neuronales convolucionales ya creadas. Se han utilizado algunas de ellas como puede ser "Resnet50" (He, K., Zhang, X., Ren, S., & Sun, J. (2016)), "InceptionV9" (*Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015)*) o "VGG16" (Simonyan, K., & Zisserman, A. (2014)) sin preentrenar, es decir, todas las capas de la red se entrenan con nuestros datos y preentrenadas, lo que implica que, en algunas de las capas, normalmente las primeras, los pesos entre las conexiones han sido marcados por otro set de datos y con el nuestro simplemente se entrenan las últimas capas modificando los valores obtenidos por la propagación de las otras capas. Esto último puede ser útil cuando se tiene un data

[10] http://www.diegocalvo.es/red-neuronal-convolucional/

set de grandes dimensiones ya que, al utilizar redes preentrenadas los tiempos de ejecución disminuyen considerablemente y los resultados siguen siendo buenos ya que las últimas capas son entrenadas con gran cantidad de datos, por tanto, con gran cantidad de información.

Es importante señalar que, a la hora de encontrar un mejor modelo, además de utilizar distintas estructuras de redes neuronales convolucionales, se han cambiado diferentes parámetros como el número de épocas o el tamaño del lote.

El número de épocas corresponde con el número de veces que pasan los datos por toda la red. Por lo que, el algoritmo aprenderá pasando los datos de entrenamiento época tras época hasta que se estabilizan los pesos y la salida de la red converge hasta un valor aceptable.

Por otro lado, el tamaño del lote corresponde al número de datos que pasa por la red en cada una de las épocas.

4.1.11. KNN (K-Nearest-Neighbor)

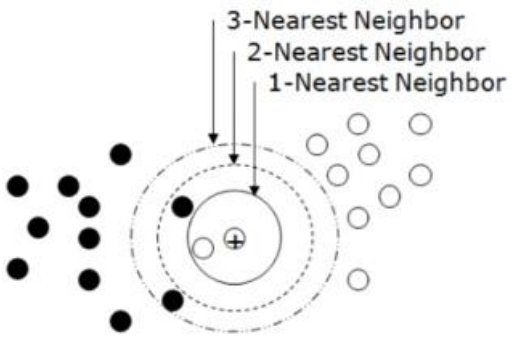
K-Nearest-Neighbor o k vecinos más cercanos es otro algoritmo de Machine Learning que también puede ser utilizado para predicción o para clasificación. Básicamente consiste en clasificar a los datos en función de los individuos más parecidos que se encuentran cerca (vecinos). Es decir, clasifica una observación en función de las observaciones que le rodean.

La manera de funcionamiento de este algoritmo consiste en tres sencillos pasos.

1. Calcular la distancia entre una observación y el resto
2. Seleccionar el número de vecinos más cercanos (k)
3. La observación en cuestión será clasificada en función a la clase que predomina en las observaciones que se encuentren a su alrededor.

A continuación, se muestra un ejemplo de “Analíticaweb.es”.

Figura 10. Ejemplo de KNN



Web Data Mining, de Bing Liu

- Para *k* = 1 el algoritmo clasificará la bola con signo + como blanca
- Para *k* = 2 el algoritmo no tiene criterio para clasificar la bola con signo +
- Para *k* >= 3 el algoritmo clasificará la bola con signo + como negra [11]

En este caso, el algoritmo de knn se ha utilizado para clasificar de una manera diferente a la que se han utilizado el resto de los algoritmos. El knn se ha utilizado para hacer una clasificación pixel by pixel con el fin de que, al pasarle una foto completa, sea capaz de clasificar que píxeles contienen mala hierba y cuáles no. Así cumplir con el objetivo de detectar donde se encuentra exactamente la mala hierba en el campo de cultivo.

Para encontrar el modelo que mejor clasifica cada uno de los píxeles se ha probado a variar el número de vecinos.

4.1.12. EVALUACIÓN DE LOS MODELOS

A la hora de evaluar y comparar los modelos realizados, se han utilizado diferentes medidas y estadísticos comunes en problemas de clasificación. La siguiente tabla representa una matriz de confusiones de la cual se puede obtener la siguiente información:

Figura 11. Matriz de confusiones teórica

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

- Tasa de aciertos: $\frac{VN+VP}{VN+FP+FN+VP}$
- Tasa de fallos: $\frac{FP+FN}{VN+FP+FN+VP}$
- Sensibilidad o tasa de verdaderos positivos (Recall): $\frac{VP}{FN+VP}$
- Especificidad o tasa de verdaderos negativos: $\frac{VN}{VN+FP}$
- Valor predictivo positivo: $\frac{VP}{FP+VP}$

[11] https://www.analiticaweb.es/algoritmo-knn-modelado-datos/

- Valor predictivo negativo: $\frac{VN}{VN+FN}$
- Accuracy (exactitud): $\frac{VP+VN}{VP+VN+FP+FN}$
- Precisión : $\frac{VP}{FP+VP}$

Además de estas medidas de adecuación, se ha utilizado para comparar los modelos la curva ROC y el área bajo dicha curva.

La curva ROC representa la sensibilidad frente a 1-especificidad (tasa de falsos positivos). Debido a que tanto la sensibilidad como la especificidad van de 0 a 1, el área bajo la curva ROC solo podrá tomar valores entre 0 y 1. Cuanto más cercano a 1 sea esta área, mejor será el modelo que estamos evaluando. Por el contrario, si esta medida de evaluación toma el valor de 0,5, será un modelo muy malo. Ya que se trataría de un modelo de clasificación totalmente aleatorio.

- **Validación cruzada**

La validación cruzada es un método mediante el cual se comparan diferentes modelos de clasificación o predicción, el fin de este es garantizar que los resultados son independientes de la partición de datos de entrenamiento y validación.

Este método consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones.

En este proyecto, se ha utilizado solamente para comparar los modelos más sencillos, como la regresión logística y las redes neuronales más simples. Si los modelos son muy complejos o están hechos con muchas observaciones, los tiempos de ejecución aumentan considerablemente. En el apartado de descripción y exploración de los datos, se explica con mayor profundidad como se han realizado las particiones de datos y de qué manera han sido comparados unos modelos con otros.

4.2. TECNOLOGÍAS UTILIZADAS

Para la realización de este estudio se han utilizado distintos softwares como son SAS y Python.

Toda la parte de tratamiento de imágenes, Deep Learning y cualquier modelo realizado directamente a través de le imagen se ha hecho con Python. Sin embargo, los modelos realizados con la media de R, de G y de B se han hecho con SAS, además de la descripción de las variables.

Cuando se trabaja con imágenes se necesita gran cantidad de memoria para procesar toda esa información ya que, una imagen, por ejemplo, de 32x32 píxeles (imagen muy pequeña) consta de una matriz de 1024 elementos con información sobre las medidas de los colores de cada uno de los píxeles. Por ello, es bastante común utilizar memorias del tipo GPU y TPU que son explicadas a continuación.

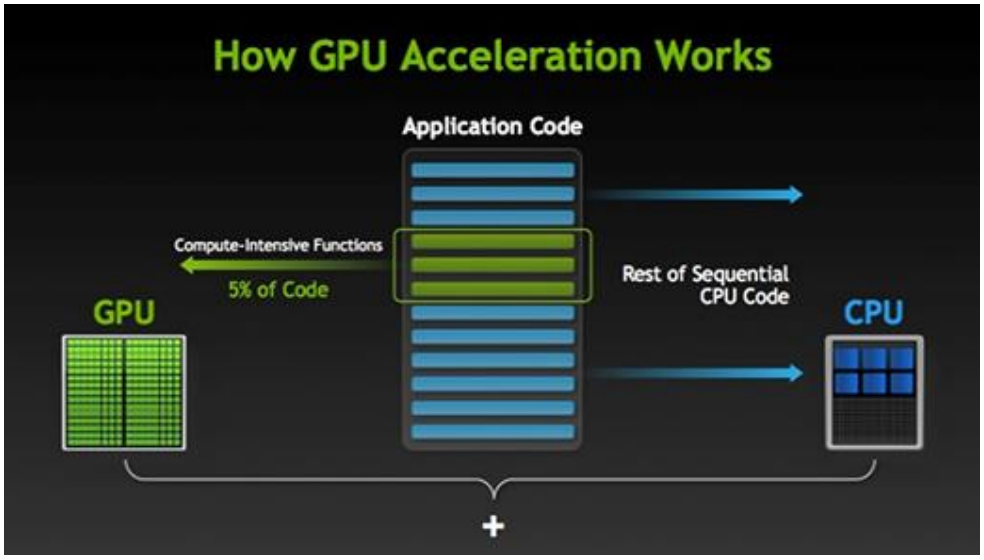
4.2.1. MEMORIA GPU

Según el diccionario de informática y tecnología "GPU significa en inglés Graphics Processing Unit, en español Unidad de Procesamiento de Gráficos. El GPU es un chip procesador, como un chip CPU (Unidad Central de Procesamiento), pero es principalmente empleado para funciones gráficas en computadoras. Estas funciones gráficas pueden ser para efectos de luz, transformaciones de objetos, animación 3D etc."[12]

Una memoria GPU consta de una unidad de procesamiento de gráficos que en su día fue creada para ser el procesador central en videojuegos, entre otras cosas. Pero, alrededor del año 2006 con la aparición de CUDA (Arquitectura de Dispositivos de Cómputo Unificado), que permite programar en GPU con lenguajes a muy alto nivel como por ejemplo C++ o Python, se comienza a utilizar la GPU para el tratamiento de imágenes ya que, son mucho más rápidos los cálculos realizados a los píxeles.

Es tan eficiente debido a que la GPU permite asignar las tareas de ejecución más intensivas para la propia GPU y el resto de las tareas para la memoria CPU. Esto hace que a la hora de ejecutar una red neuronal convolucional el proceso sea mucho más rápido en GPU que en CPU. La figura 12 resume dicha asignación de tareas.

Figura 12. Asignación de tareas en GPU



[13]

[12] http://www.alegsa.com.ar/Dic/gpu.php
[13] https://la.nvidia.com/object/what-is-gpu-computing-la.html

Debido al impulso que ha provocado el aprendizaje profundo en el tratamiento de imágenes, muchas plataformas de Machine Learning en la nube como Google Cloud Platform o Amazon Cloud, ofrecen memoria GPU para que los científicos de datos puedan ejecutar sus modelos de una forma ágil. Obviamente estos servicios que ofrecen este tipo de compañías son de pago, aunque, para realizar este trabajo se ha utilizado el espacio que ofrece Google en Drive "Google Collab".

4.2.2. GOOGLE COLLABORATORY

Se trata de una herramienta dentro de Google Drive que permite realizar Jupyter Notebooks[14] sin configuración previa (casi todas las librerías vienen ya definidas y no hace falta instalarlas) y además se ejecuta en la nube.

Una de las grandes ventajas que ofrece Google Collab es que permite ejecutar códigos complejos de Machine Learning en GPU de forma gratuita.

Esta herramienta también ofrece gratuitamente unidades TPU que son unidades de procesamiento de Tensor. Que agilizan mucho la ejecución de procesos de aprendizaje profundo, hasta más que la unidad GPU.

Para el estudio se ha utilizado Google Collaboratory prácticamente siempre que se ha utilizado Python para construir un modelo de Machine Learning. Además, todos los modelos han sido ejecutados en GPU, sin embargo, la TPU no ha sido utilizada.

4.2.3. ALGUNAS LIBRERÍAS DE PYTHON INTERESANTES

- **Procesamiento de imágenes:**

  Se han utilizado librerías como **Pillow** y **OpenCV** a la hora de procesar la imagen. Es decir, para hacer labores como recortar las imágenes o redimensionarlas (ensancharlas o estrecharlas cambiando su número de píxeles).
  Estas librerías contienen cantidad de opciones a la hora de trabajar con imágenes desde acciones básicas como leer la imagen hasta detección de bordes o colores dentro de una imagen (esto último no se ha realizado en este proyecto).

- **Machine Learning:**

  Una de las librerías de Python con más opciones a la hora de hacer modelos de Machine Learning es la librería **sklearn**. En este caso se ha utilizado para hacer el estudio del knn píxel by pixel a través de la opción "KNeighborsClassifier".
  Esta librería también contiene funciones tan útiles como aquella que divide muestras aleatoriamente en entrenamiento y validación y otras como la opción

[14] El Jupyter Notebook es un entorno interactivo web de ejecución de código en los que, por ejemplo, puedes incluir gráficas que ayuden en el análisis y explicación de tus datos. Utilizados para facilitar la explicación y reproducción de estudios y análisis

"metrics" que permite conseguir de manera sencilla métricas como la precisión o la sensibilidad.

- **Aprendizaje profundo:**

  A la hora de lanzar modelos de Deep Learning, se ha utilizado la librería **keras**, esta librería corre por encima de **Tensorflow** que consiste en un lenguaje de programación muy por debajo, por lo que keras, hace mucho más fácil el uso del Deep Learning.
  Keras contiene implementadas gran cantidad de arquitecturas de redes neuronales convolucionales como ResNet o VGG. Además, también te permite crear tu propia estructura de red neuronal (convolucional o no). Por otra parte, también tiene implementados otros set de datos para poder lanzar modelos preentrenados.

5. DESARROLLO DEL TRABAJO Y PRINCIPALES RESULTADOS

Dado que en la metodología ya se ha explicado la recogida de los datos, el primer punto para desarrollar el proyecto es obtener las sub-imágenes. Después se hará un análisis descriptivo de las características de los rectángulos y se comenzará con los análisis de clasificación por imagen y por píxel.

5.1. OBTENCIÓN DE LAS SUB-IMÁGENES

Para obtener cada una de las sub-imágenes como imagen independiente, se ha desarrollado un bucle en lenguaje Python que recorre el Excel inicial de manera que, entraría en la columna Carpera y se metería en una carpeta creada con el nombre de cada uno de los identificadores de las carpetas. Dentro de cada carpeta se obtienen las imágenes de un mismo día. Una vez dentro de esa carpeta el bucle recorrería el Id de imagen y obtendrá la información de cada uno de los rectángulos de cada imagen. Después, se utilizará la función de Python "crop" de la librería Pillow para recortar las imágenes con las coordenadas marcadas en las variables "fila1", "col1", "fila2", "col2". El código utilizado es el siguiente:

Figura 13. Fragmento de código para obtener las sub-imágenes

```
cont = 0
for indx,row in data_img.iterrows():
    car=row['Carpeta']
    carp = str(car)
    dir_img = '/home/cxiome/Escritorio/TFM/workspace/carpetas_antiguas/' + carp
    #for filename in os.listdir(dir_img):
    image = str(row['imagen'])
    image_str = dir_img + '/IMG_' + image + '.JPG'
    img = Image.open(image_str)
    fila1_int = int(row['fila1'])
    col1_int = int(row['col1'])
    fila2_int = int(row['fila2'])
    col2_int = int(row['col2'])
    box = (col1_int, fila1_int, col2_int, fila2_int)
    cropped_image = img.crop(box)
    cont = cont + 1
    cont_str = str(cont)
    name = "/home/cxiome/Escritorio/TFM/workspace/tests/" + cont_str
    cropped_image.save(name + ".JPG")
```

El bucle anterior fue realizado en el sistema operativo Linux ya que es mucho más fácil mover archivos y trabajar con carpetas desde código Python que en Windows.

Después de realizar el bucle que recorta las imágenes, se obtienen 1894 rectángulos de diferentes tamaños. Esto se debe a que los rectángulos donde el experto definió si había malezas o no, tenían un tamaño diferente. En las siguientes figuras, se puede ver el aspecto que tienen las sub-imágenes con mala hierba y sin mala hierba.

Figura 14. Ejemplos rectángulos sin mala hierba

Figura 15. Ejemplos rectángulos con mala hierba

La figura 14 contiene varias sub-imágenes que el experto clasificó como cultivos de cereales sanos y en la figura 15 se encuentran algunas imágenes que si contienen mala hierba. Como ya se ha comentado anteriormente, las sub-imágenes con mala hierba tienen un tono más verde. Además, se puede ver como las imágenes contienen tamaños diferentes en las distintas clases.

Ahoa que se tienen las sub-imágenes como imágenes independientes, se pueden sacar características que describan a cada una de ellas. Las variables que se van a extraer de

cada uno de los píxeles de las sub-imágenes serán los canales de color RGB. Obteniéndose una media de cada uno de los tres canales para cada uno de los rectángulos. De esta manera, se podrán hacer análisis de clasificación de las sub-imágenes a partir de variables numéricas, como son los modelos de regresión logística y los de redes neuronales sencillas que serán construidos en SAS.

En el momento en el que se tienen todos los datos, se puede comenzar a hacer los modelos de clasificación, pero antes se va a resumir brevemente la información que se tiene hasta el momento.

Para realizar los modelos de clasificación, finalmente, contamos con una base de datos de 1894 imágenes, de las cuales, se ha hecho una división aleatoria para entrenamiento y test. Concretamente, se han utilizado un 97% de los rectángulos para entrenamiento (que más tarde se dividirán entre entrenamiento y validación) y un 3% para testear el modelo conseguido con imágenes que no han pasado por el modelo. Además, dentro de los modelos de clasificación realizados con Python, que son los de redes neuronales planas y los de aprendizaje profundo, no se va a llevar a cabo validación cruzada, ya que, requiere mucho tiempo de ejecución y no es un procedimiento común cuando se utilizan algoritmos tan complejos. Sin embargo, entre los modelos realizados en SAS (regresión logística y redes aún más simples) sí que se usará la validación cruzada para las comparaciones entre esos modelos y el mejor modelo resultante será cotejado con los realizados en Python utilizando la muestra de test separada al principio.

5.2. DESCRIPCIÓN Y EXPLORACIÓN DE LAS SUB-IMÁGENES

Antes de desarrollar las labores necesarias para efectuar los objetivos del proyecto, se va a indagar en la naturaleza de los datos que se tienen de las medias de los tres canales de color RGB de cada uno de los rectángulos. Se mostrarán diferentes tablas y gráficos con información del número de rectángulos de cada clase, valores de los estadísticos descriptivos básicos de las medias de los canales RGB y la distribución de cada uno de los canales.

Para empezar, se ha calculado la frecuencia de imágenes que tienen mala hierba (codificado como 1) y que no la tienen (codificado como 0).

Tabla 2. Frecuencia de clada clase

Mala Hierba (1)	Sin mala hierba (0)
1194	700

Como se puede ver en la tabla 2, se obtienen 1194 rectángulos con mala hierba y los 685 restantes sin mala hierba. A continuación, se han obtenido las variables Rmedia, Gmedia y Bmedia de cada uno de los rectángulos. De esta manera se ha modelado cada imagen de manera sencilla en base a estas tres variables. La tabla 3 muestra algunos estadísticos descriptivos.

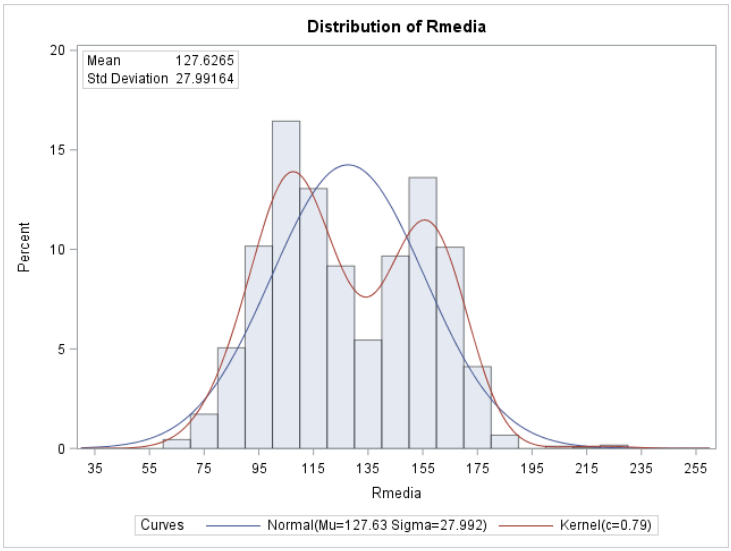
Tabla 3. Algunos estadísticos descriptivos de las variables

Variable	Label	Maximum	Minimum	Mean	Std Dev	Mode	Lower Quartile	Median	Upper Quartile
Rmedia	Rmedia	228.5042064	61.7357143	127.6265429	27.9916389	113.7428571	104.8275606	122.5914043	153.3494787
Gmedia	Gmedia	217.8280577	67.1616071	127.8080059	20.5113046	.	113.2635877	126.5734978	142.5914483
Bmedia	Bmedia	185.4226036	36.1142857	86.6078190	20.5947249	72.9444444	69.8103600	85.5369469	103.0432824

Tal y como indica la tabla 3, la media del color azul es la que toma colores más bajos, esto ocurre debido a que, en las imágenes del estudio, el color azul no interviene en ningún momento. El color verde, que es el color predominante en las sub-imágenes con mala hierba, obtiene la media y mediana más alta, seguida muy de cerca por la media del color rojo, que también obtiene valores altos tanto en media como en mediana, además, el máximo del color rojo es superior al del color verde.

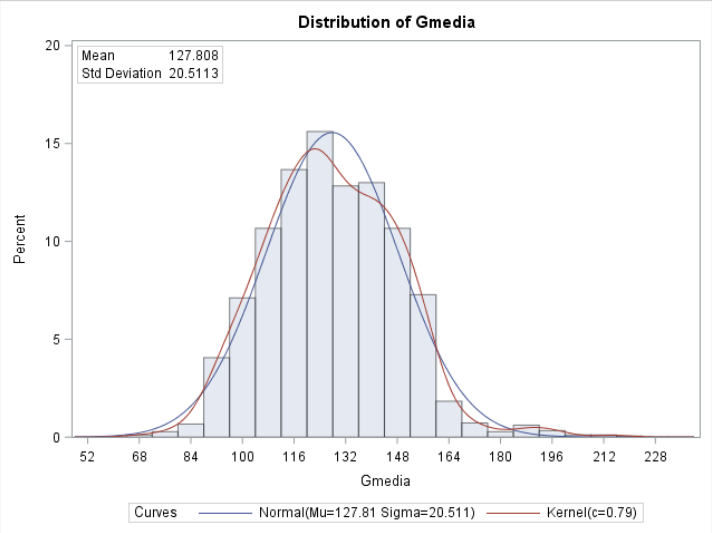
Para terminar de entender por completo el conjunto de sub-imágenes, las figuras 16, 17 y 18 son una representación de las distribuciones de las medias de rojo, verde y azul.

Figura 16. Distribución de la media del canal rojo



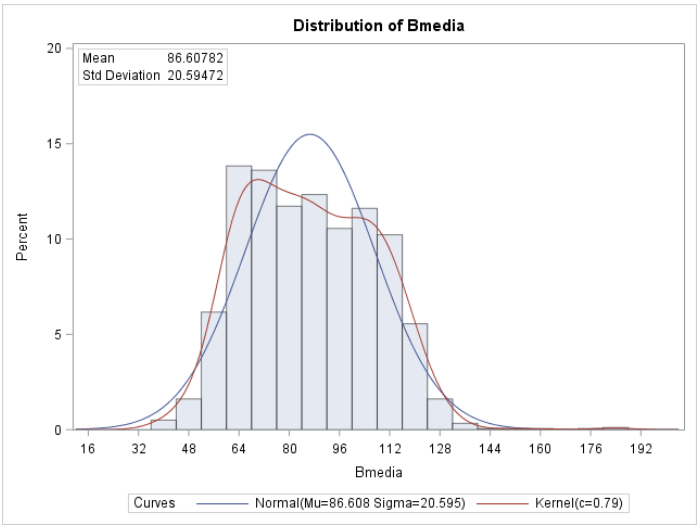
En la figura 16 se puede ver la distribución de la media del canal rojo en las sub-imágenes, se trata de una distribución bimodal que toma dos valores máximos en frecuencia, uno en 100 y otro en 155. Puede deberse a los distintos valores medios que contienen los rectángulos en cada una de las clases representadas. Además, la media se encuentra en 127.63 con una desviación típica de 27.992.

Figura 17. Distribución de la media del canal verde



La figura 17 hace referencia a la distribución de la media del canal verde, esta se asemeja a una distribución normal cuya moda se encuentra alrededor de 120. La media toma un valor de 17.81 con una desviación típica de 50.511.

Figura 18. Distribución de la media del cana Azul



La distribución de la media del canal azul en la muestra de sub-imágenes también se asemeja a una normal cuyo máximo se encuentra en 64. La media del canal azul toma un valor de 86.608 con una desviación típica de 20.59.

También es interesante ver un descriptivo de las variables diferenciado por las clases, es decir, como se distribuyen las variables y cuáles son sus estadísticos descriptivos cuando se tiene mala hierba y cuando no. En primer lugar se obtienen los estadísticos

descriptivos más básicos para cada una de las clases. La figura 19 hace referencia a imágenes sin mala hierba y la 20 a imágenes con mala hierba.

Figura 19. Estadísticos descriptivos para los rectángulos sin mala hierba

Variable	Etiqueta	Máximo	Mínimo	Media	Dev std	Moda	Cuartil inferior	Mediana	Cuantil superior
Rmedia	Rmedia	228.5042064	124.7469758	156.6331080	12.6763456	.	148.5670270	156.8514216	164.5975904
Gmedia	Gmedia	217.8280577	99.7596041	143.1251478	13.2370769	.	135.4235096	143.3402238	151.0471057
Bmedia	Bmedia	185.4226036	57.6034091	105.6047229	13.8544850	.	98.5475024	106.1588641	113.7672083

Figura 20. Estadísticos descriptivos para los rectángulos con mala hierba

Variable	Etiqueta	Máximo	Mínimo	Media	Dev std	Moda	Cuartil inferior	Mediana	Cuantil superior
Rmedia	Rmedia	182.6420455	61.7357143	110.4279600	18.9190409	113.7428571	98.5504470	108.2579954	120.0727273
Gmedia	Gmedia	200.1444444	67.1616071	118.7261606	18.5680286	.	107.2173913	117.5898874	126.7517361
Bmedia	Bmedia	132.2307692	36.1142857	75.3441681	14.8647947	72.9444444	64.2355612	73.9326274	85.1452381

Como se puede ver, casi todos los estadísticos referentes a las medias de los tres canales toman valores bastante más altos cuando la imagen no contiene mala hierba. Por otro lado, las siguientes figuras representan la distribución de los tres canales de color para cada clase.

Figura 21. Distribución de la media del canal rojo cuando no hay mala hierba

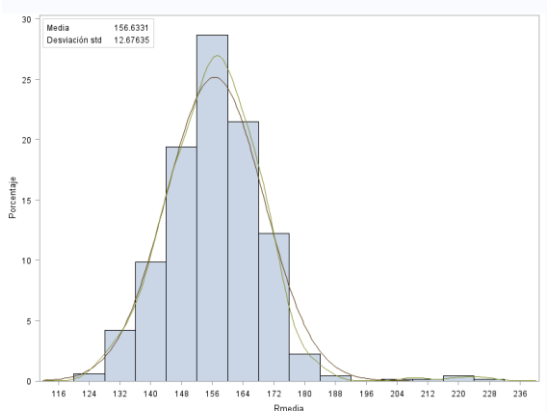


Figura 22. Distribución de la media del canal rojo cuando hay mala hierba

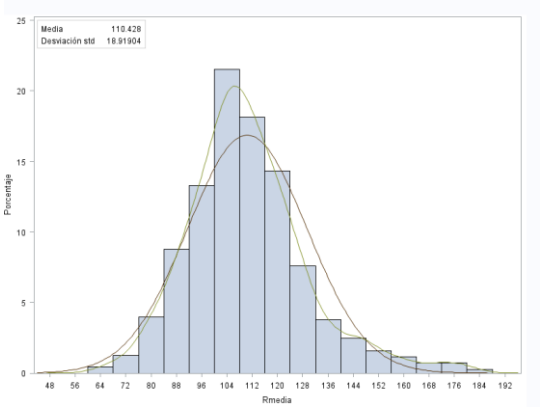


Figura 23. Distribución de la media del canal verde cuando no hay mala hierba

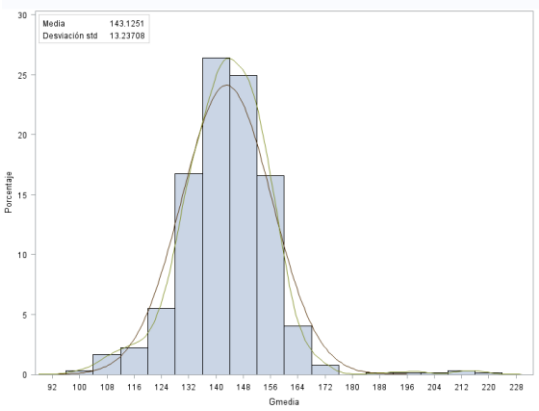


Figura 24. Distribución de la media del canal verde cuando hay mala hierba

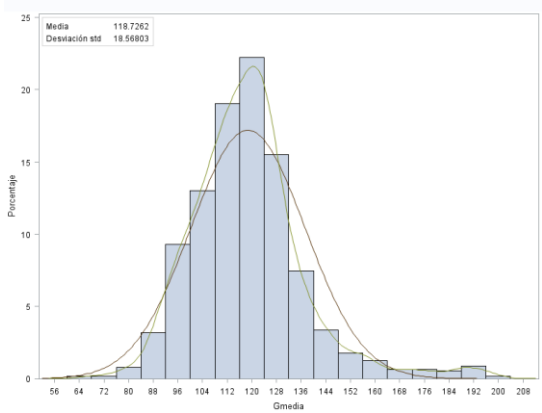


Figura 25. Distribución de la media del canal azul cuando no hay mala hierba

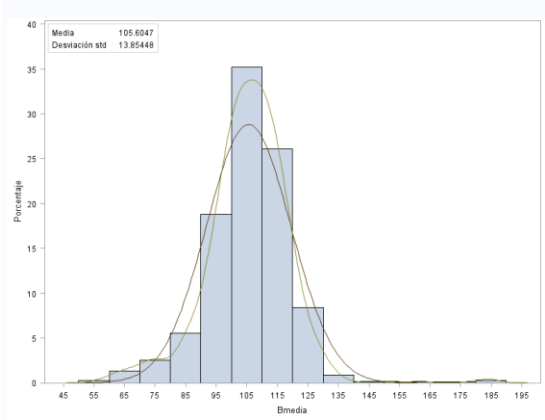
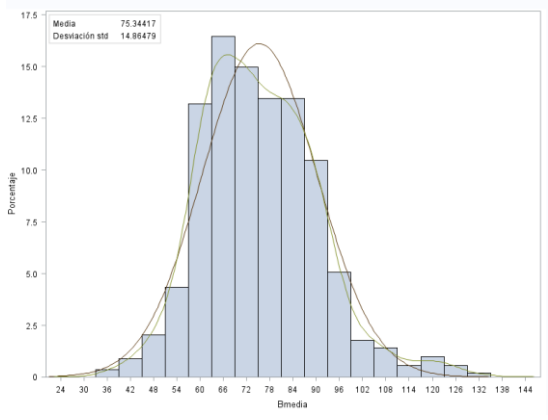


Figura 26. Distribución de la media del canal azul cuando hay mala hierba



Parece que todas las medias de los colores RGB para las distintas clases siguen una distribución normal. Aunque, de la misma manera que las tablas de estadísticos descriptivos, las sub-imágenes toman valores diferentes por clase en las tres variables y las distribuciones son distintas.

5.3. MODELO DE CLASIFICACIÓN DE SUB-IMÁGENES

Bajo este apartado, se va a realizar la búsqueda de un mejor modelo que clasifique cada una de las sub-imágenes en mala hierba (1) o no mala hierba (0). Para ello se han empleado métodos como la regresión logística, redes neuronales simples, redes neuronales complejas y Deep Learning.

Para empezar, se van a designar los resultados de las **redes neuronales complejas**, es decir aquellas que contienen más de una capa oculta (en el caso de este trabajo, de una a tres capas y varios nodos en cada una de ellas) realizadas con keras, después se probará cómo funciona la clasificación con Deep Learning, estos modelos se van a realizar a través del valor de los píxeles en bruto y, por último, con el software SAS, se van a hacer modelos de redes neuronales aún más simples (con tan solo una capa oculta y varios nodos en ella) y de regresión logística, estos últimos modelos serán entrenados con las medias de RGB en cada uno de los rectángulos. Con ello se comprobará si en una base de datos con pocas clases y pocas imágenes merece la pena hacer modelos tan complejos.

5.3.1. REDES NEURONALES COMPLEJAS

Cabe destacar que, la hora de hacer las redes neuronales complejas con la librería keras, se han hecho varios pasos previos antes de empezar a modelizar.

Para comenzar se debe leer el Excel con los datos de entrenamiento para tener en una tabla el id de cada imagen con su etiqueta. Después se ha hecho un bucle que recorra todas las imágenes de entrenamiento y las redimensione para transformarlas en un array. Este redimensionamiento se hace para que todas las imágenes tengan el mismo

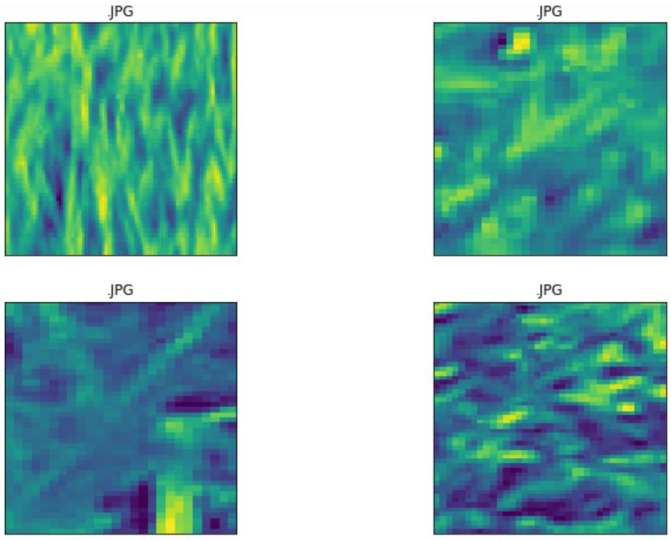
tamaño ya que es así como deben entrar al modelo. Las imágenes se han redimensionado a 224 píxeles debido a que en modelos más complejos el tamaño de las imágenes debe ser al menos de este tamaño para poder entrar en la red (tener imágenes tan grandes hace que los tiempos de ejecución sean más altos, al disponer de GPU en Google Collaboratory se puede hacer, pero seguramente, si se ejecutase en local se obtendría un error de memoria). Al hacer esto, como había imágenes de varios tamaños diferentes, algunas se han ensanchado y otras se han estrechado. El siguiente paso es estandarizar los píxeles para que todos tomen valores entre 0 y 1. Esto hace que el modelo prediga mejor al igual que en un modelo de datos convencional. Una vez hecho todo esto se puede comenzar a crear el modelo. La figura 27, presenta el código utilizado para introducir las imágenes en el array y redimensionarlas.

Figura 27. Fragmento de código para obtener las imágenes en el formato deseado

```
#define a function to prepare our trianing images
def PrepareTrainImages(dataframe, shape):

    #obtain the numpy array filled with zeros having the format --> (batch_size, height, width, channels)
    #x_train = np.zeros((shape, 100, 100, 3))
    count = 0

    for fig in dataframe['image_id']:

        fig_str = str(fig)
        #load images into images of size 100x100x3
        img = load_img("drive/My Drive/tfm/samples/train_data/" + fig_str, target_size = (224, 224, 3))

        #convert images to array
        x = img_to_array(img)
        x = preprocess_input(x)

        x_train[count] = x
        count += 1

    return x_train

x_train = np.zeros((df_img.shape[0], 224, 224, 3))
x_train = PrepareTrainImages(df_img,df_img.shape[0])
```

La figura 28, exhibe como se ven las imágenes después de hacer el redimensionamiento.

Figura 28. Ejemplo de las imágenes redimensionadas.



Después de preparar las imágenes para que entren en el modelo, se necesita fabricar la red, para ello se ha empleado la función “Sequential” de la librería keras. Seguidamente, se manifiesta como se constituye una red de dos capas ocultas con 100 y 50 nodos en cada capa.

En primer lugar, se define la estructura de la red. (Figura 29). Para ello, se añade cada una de las capas (la de entrada, la de salida y las dos ocultas). después se llama a la función con la estructura de la red y se indica cual es la función de pérdida que se quiere usar, en este caso, se usará la de entropía cruzada.

Figura 29. Fragmento de código de definición de estructura de la red

```
def create_simple_nn():
  model = Sequential()
  model.add(Flatten(input_shape=(224, 224, 3), name="Input_layer"))
  model.add(Dense(100, activation='relu', name="Hidden_layer_1"))
  model.add(Dense(50, activation='relu', name="Hidden_layer_2"))
  model.add(Dense(2, activation='softmax', name="Output_layer"))

  return model

snn_model = create_simple_nn()
snn_model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['acc', 'mse'])
```

Se puede ver un resumen de la red utilizando la función “summary”, en este caso el resumen quedaría:

Figura 30. Resumen de la red con dos capas ocultas y 100 y 50 nodos en cada capa

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_layer (Flatten)        (None, 150528)            0
_________________________________________________________________
Hidden_layer_1 (Dense)       (None, 100)               15052900
_________________________________________________________________
Hidden_layer_2 (Dense)       (None, 50)                5050
_________________________________________________________________
Output_layer (Dense)         (None, 2)                 102
=================================================================
Total params: 15,058,052
Trainable params: 15,058,052
Non-trainable params: 0
_________________________________________________________________
```

Como se puede ver en la figura 19, la primera capa oculta contiene 150052900 parámetros, la segunda 5050 y la capa de salida 102 (la capa de salida contiene 2 nodos porque se tienen dos categorías para la variable respuesta).

Por último, se especifican las épocas, el tamaño del lote, las muestras de entrenamiento y validación y el objeto que guarda las etiquetas de las imágenes.

Figura 31. Fragmento de código para entrenar la red

```
snn = snn_model.fit(x=x_train, y=y_train, batch_size=16, epochs=10, validation_data=(x_test, y_test))
```

Es preciso que se han realizado varios modelos de redes neuronales simples con la librería keras cambiando diferentes parámetros como el número de capas ocultas, el número de nodos en cada una de las capas, el tamaño del lote y el número de épocas. Téngase en cuenta que, las redes realizadas con keras siempre se han construidos con la función de activación "ReLu" en las capas ocultas y SoftMax en la capa de salida en todo el estudio. Todos los modelos han sido realizados con la misma muestra aleatoria de entrenamiento y validación (0.75 % para entrenamiento y 0,25 para validación). Además, se ha sacado el valor del área bajo la curva en el conjunto de test separado al principio para todos los modelos.

Tabla 4. Modelos de Redes neuronales

Modelo	N Capas	N de nodos por capa	Batch size	Epochs	AUC train	AUC validation	AUC Test
1	1	1000;	16	10	1	0,96	0,988
2	2	1000; 500	16	10	1	0,95	0,985
3	2	100; 50	16	10	1	0,95	0,983
4	3	1000; 500; 250	16	10	1	0,95	0,992
5	2	50;25	32	10	1	0,95	0,986
6	3	50;25;15	64	15	1	0,96	0,984
7	3	50;25;15	32	15	1	0,94	0,988
8	3	50;25;15	32	20	1	0,95	0,989
9	2	10;5	16	10	0,99	0,94	0,988
10	1	10;	16	10	0,97	0,88	0,982

Como se puede observar en la tabla 4, el modelo 4 es el que obtiene un área bajo la curva ROC más alta, el modelo 4 contiene 3 capas ocultas con 1000 nodos en la primera capa, 500 en la segunda y 250 en la tercera, un tamaño del lote de 16 y 10 iteraciones. A continuación, se pueden observar las curvas ROC para el conjunto de entrenamiento y validación (figuras 32 y 33).

Figura 32. Curva ROC en Train, Red Neuronal

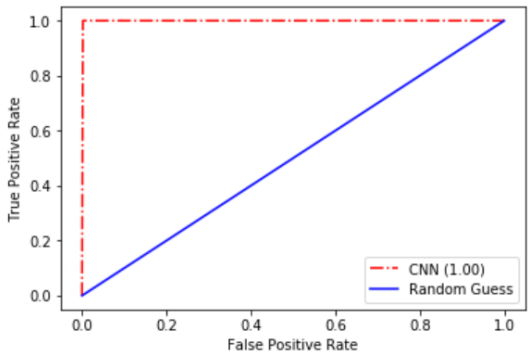
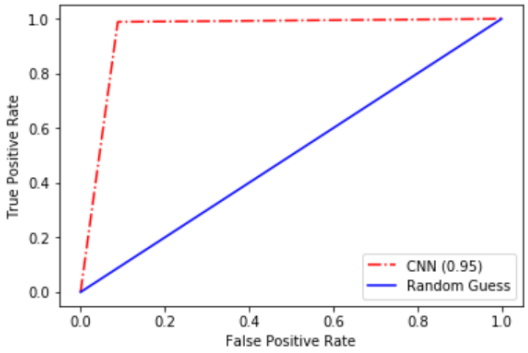


Figura 33. Curva ROC en Test, Red Neuronal



La figura 32 muestra la curva ROC en el conjunto de entrenamiento y la figura 33 en el conjunto de validación. Las dos obtienen valores muy altos, esto puede deberse a que la clasificación se hace bastante obvia, ya que simplemente se necesita diferenciar el color para saber a qué clase pertenece cada imagen por color. Como se ha visto antes, los colores más verdes pertenecen a trozos con mala hierba y los más marrones a trozos en buen estado de cultivo.

5.3.2. REDES NEURONALES CONVOLUCIONALES

Aunque los resultados de las redes simples han sido muy buenos, se van a comparar diferentes modelos de **aprendizaje profundo** también realizados con la librería Keras.

Antes de empezar a hacer los modelos, se han tenido que seguir los mismos pasos que al hacer las redes neuronales simples (redimensionar las imágenes, estandarizar los píxeles, convertir las imágenes en array, etc).

En este punto, las redes neuronales convolucionales que se han utilizado son redes con una estructura ya definida que Keras pone a disposición de cualquier usuario. Las empleadas en este apartado del proyecto son la VGG19, la InceptionV9 y la Resnet50. Todas ellas son estructuras muy complejas con gran cantidad de capas ocultas y convolucionales. Dado que no se construye la red como en el caso del apartado anterior, el código referido a elaborar el modelo cambia, ahora simplemente se importa la arquitectura deseada y se le indica cuantas capas se quieren entrenar (en caso de no querer entrenarlas todas), las funciones de activación a emplear, la función de pérdida, la tasa de aprendizaje, el número de épocas y el tamaño del lote.

En seguida, son exhibidos los diferentes modelos de redes neuronales convolucionales probados en la búsqueda de un mejor clasificador de imágenes. Cabe destacar que en un primer momento solo fueron entrenadas las últimas capas de estas estructuras, dejando el resto de las capas congeladas. También se ha probado a entrenar la red entera. Además, se le han cambiado los diferentes parámetros tal y como muestra la siguiente tabla resumen.

Tabla 5. Modelos de aprendizaje profundo

Modelo	Arquitectura	Capas Train	Batch size	Epochs	AUC Train	AUC validation	AUC Test
1	ResNet50	5	16	5	0,5	0,5	0,747
2	ResNet50	30	32	20	0,55	0,52	0,808
3	VGG19	25	32	20	0,58	0,52	0,81
4	InceptionV9	40	32	15	0,61	0,66	0,84
5	VGG19	Todas	16	5	0,98	0,98	0,998
6	InceptionV9	Todas	16	5	0,98	0,96	0,993
7	VGG16	todas	16	5	0,99	0,98	0,998
8	ResNet50	Todas	16	5	1	1	1

Tal y como muestra la tabla 5, las redes realizadas entrenando solo algunas capas con nuestros datos, dan muy malos resultados. Esto se debe a que no se tienen suficientes imágenes para que el modelo sea capaz de aprender pasando tan pocas observaciones solo por sus últimas capas (aprende más del set con el que están entrenadas las otras capas que de este). Sin embargo, En las redes convolucionales entrenadas desde 0, se vuelven a obtener valores muy altos en la evaluación.

El modelo de aprendizaje profundo que mejor clasifica los rectángulos es la ResNet50, que obtiene un valor para el área bajo la curva de 1 en entrenamiento, en validación y en test.

Figura 34. Curva de ROC, ResNet50

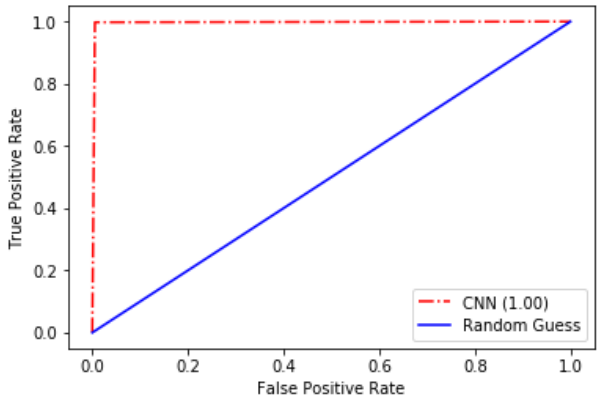
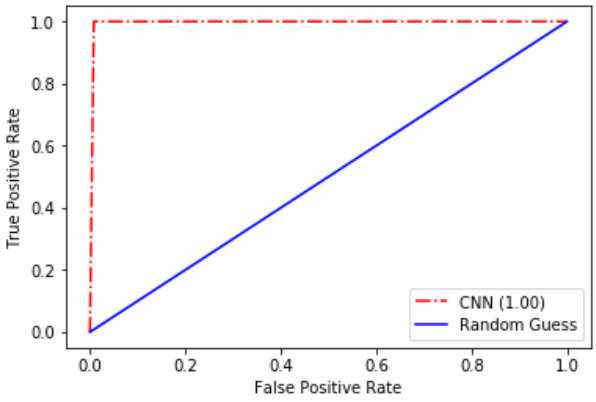


Figura 35. Curva de ROC, ResNet50

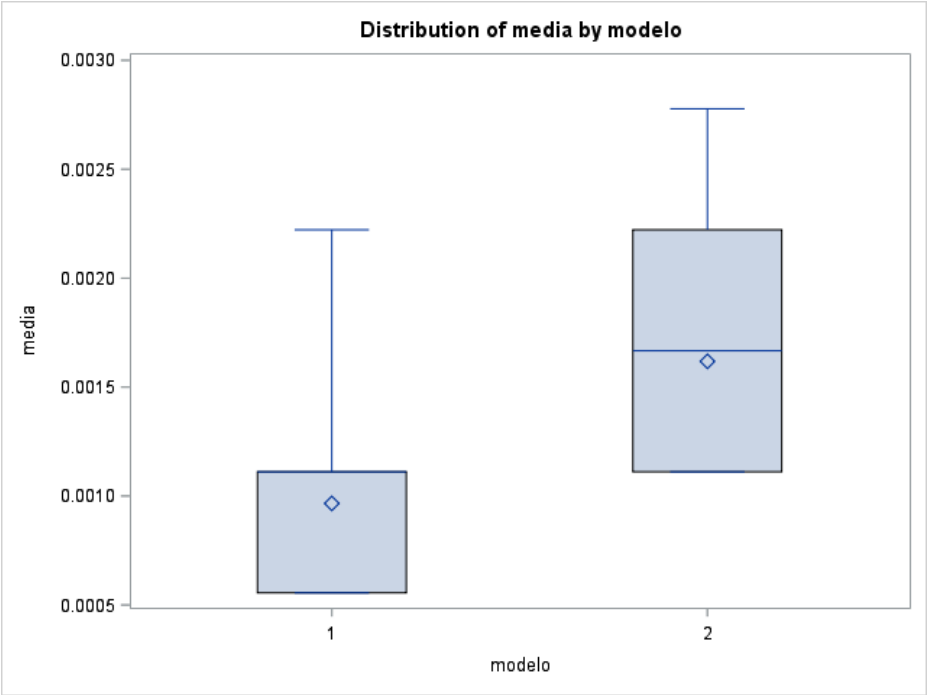


En un último paso en la parte de clasificación de las sub-imágenes, se van a probar modelos más sencillos para comprobar que, al tener pocas imágenes y clases, no merece la pena hacer modelos tan complejos para hacer una clasificación tan obvia, sino que, con modelos más sencillos se puede llegar a un buen resultado. Tal y como dicta uno de los objetivos secundarios del proyecto.

5.3.3. REGRESIÓN LOGÍSTICA

Para comenzar, se realizan dos modelos de **regresión logística** en SAS, uno con solamente las variables y otro con las variables y sus interacciones. Se recuerda que las variables para introducir al modelo son los valores de las medias de los canales rojo, verde y azul de cada sub-imagen. Como ya se ha comentado anteriormente, se ha utilizado validación cruzada para comparar los modelos construidos con el software SAS. La medida de evaluación utilizada es la tasa de fallos.

Figura 36. Modelos de regresión logística



La figura 36 consta de los resultados de la validación cruzada para el modelo que solamente contiene las variables originales (modelo 1) y el modelo que contiene también las interacciones de las variables (modelo 2).

Ambos modelos obtienen una tasa de fallos muy baja, pero el modelo 1 tiene una tasa de fallos menor que el modelo 2, por lo que se continuará realizando el estudio con las variables originales sin tener en cuenta las interacciones. Y el modelo 1 será considerado el mejor modelo obtenido en regresión logística.

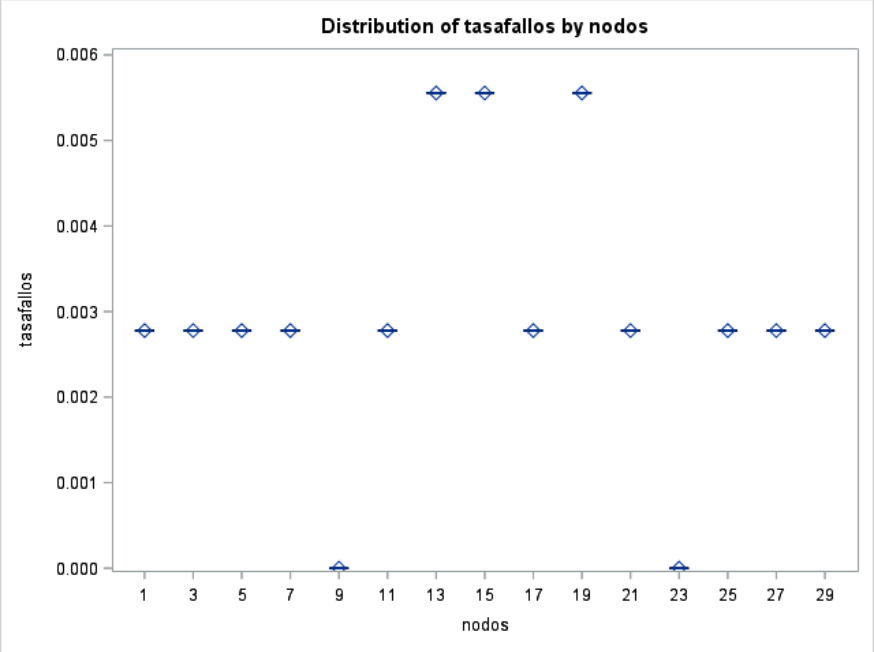
Nótese que no se ha hecho ningún método de selección de variables porque solo se consideran 3 variables a la hora de hacer los modelos y son tres variables muy influyentes en la clasificación de la variable respuesta.

5.3.4. REDES NEURONALES SIMPLES

En segundo lugar y también con el software SAS, se van a realizar diferentes modelos de **redes neuronales simples**. Primero, se va a elegir el número de nodos que mejor le funcione a la red neuronal.

Para ello se hace una validación cruzada haciendo modelos de redes neuronales desde 1 nodo hasta 37 iterando de 2 en 2 y dejando el resto de los parámetros constantes.

Figura 37. Elección del número de nodos (Redes neuronales simples)



Como se puede observar en la figura 37, con 9 y con 23 nodos se obtiene una tasa de fallos nula, sin embargo son modelos algo complejos, con un solo nodo se obtiene una tasa de fallos cercana a 0.03 y es un modelo bastante más sencillo, por lo que, se va a continuar con la búsqueda de la red que mejor clasifique probando diferentes parámetros con 1 y con 9 nodos.

A continuación, se va a comprobar que algoritmo de optimización le conviene a la red. Se van a comparar los distintos algoritmos que ofrece SAS en el procedimiento neural con 1 y 9 nodos.

Figura 38. Elección del algoritmo de optimización 1 nodo

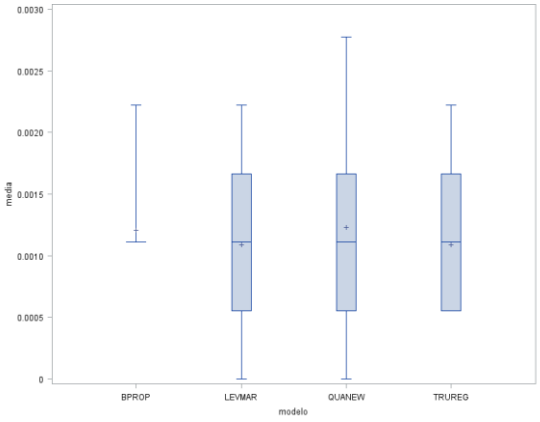
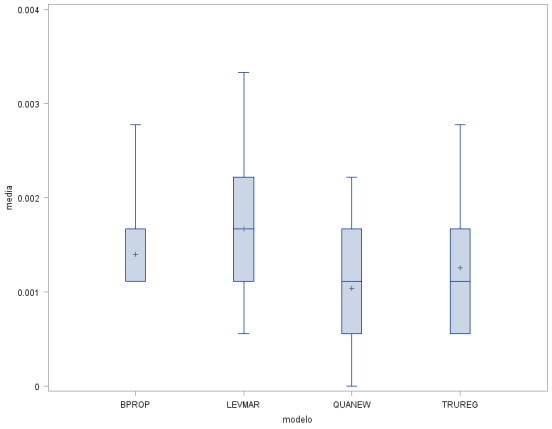


Figura 39. Elección del algoritmo de optimización 9 nodos



En la figura 38, se ve que, con un nodo el algoritmo de optimización de back propagation, es el que obtiene una menor variabilidad para la tasa de fallos en la validación cruzada. Aunque también se probará cómo funciona el modelo con el algoritmo Trureg un 1 solo nodo.

Si se mira la figura 39, no cabe duda de que el algoritmo que ofrece unos mejores resultados para 9 nodos es QUANEW.

Para terminar, se va a elegir la función de activación más eficiente comprobando back propagation con 1 solo nodo, Truhreg con un 1 nodo y QUANEW con 9 nodo.

Figura 40. Función de activación (1 nodo y backpropagation)

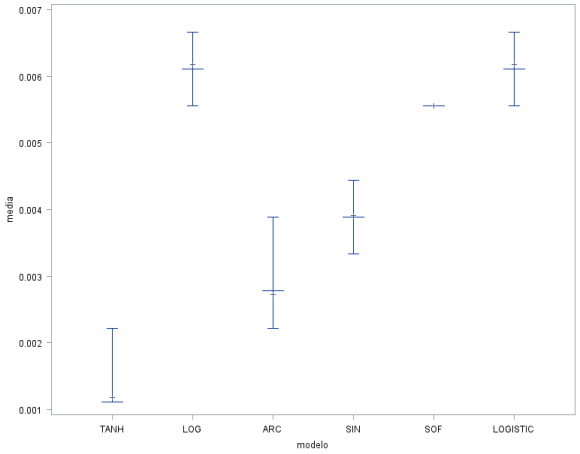


Figura 41. Función de activación(1 nodo y truhreg)

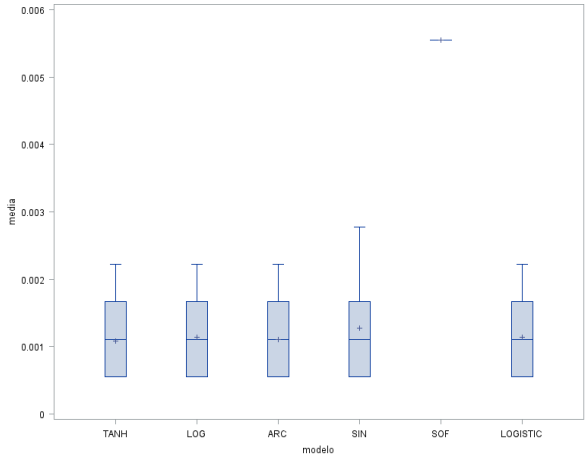
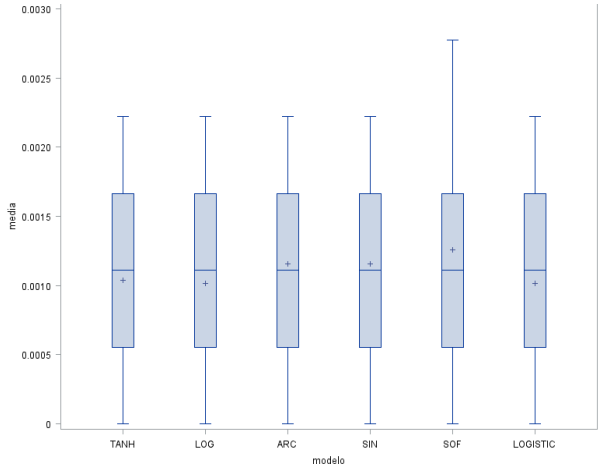


Figura 42. Función de activación (9 nodos y Quanew)



La figura 40, indica que el modelo que obtiene una tasa de fallos menor con 1 nodo y utlizando back propagation es el realizado con la función de activación de la tangente hiperbólica.

La figura 41, muestra que con un nodo y con el algoritmo de optimización de trureg, se obtienen modelos muy buenos utilizando las funciones de activación de la tangente hiperbólica, la logarítmica, la arcotangente, la del seno y la logística.

En la figura 42, se puede ver que, con 9 nodos y con el algoritmo de optimización de qwuanew también son buenas las funciones de activación: tangente hiperbólica, logarítmica, arcotangente, seno y logística.

Por todo lo explicado anteriormente, se ha hecho una tabla resumen (tabla 6) de los modelos de redes neuronales que se van a comparar definitivamente.

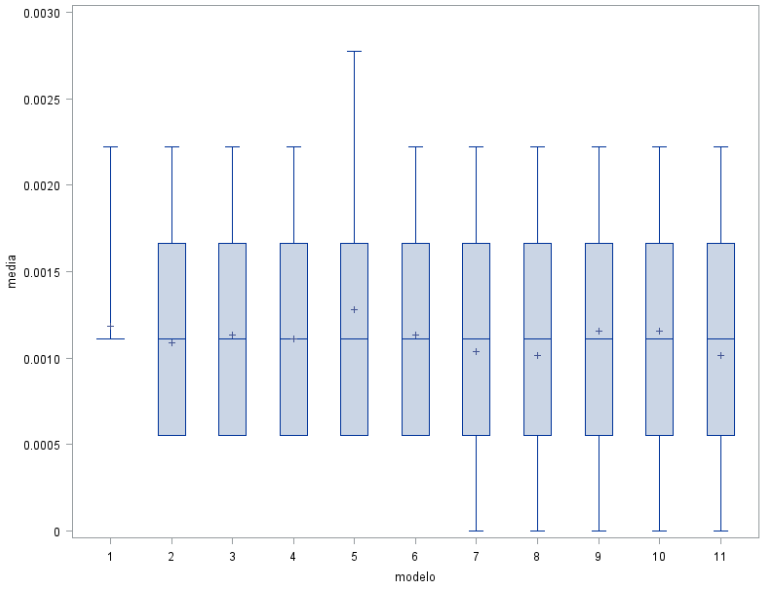
Tabla 6. Modelos de redes neuronales simples a comparar

Modelo	**Nodos**	**Algoritmo**	**F. de Activación**
1	1	bprop	tanh
2	1	trureg	tanh
3	1	trureg	log
4	1	trureg	arc
5	1	trureg	sin
6	1	trureg	logistic
7	9	quanew	tanh
8	9	quanew	log
9	9	quanew	arc
10	9	quanew	sin
11	9	quanew	logistic

La figura 43 muestra la comparación de los modelos indicados en la tabla anterior mediante validación cruzada. El modelo que presente una tasa de fallos más baja será

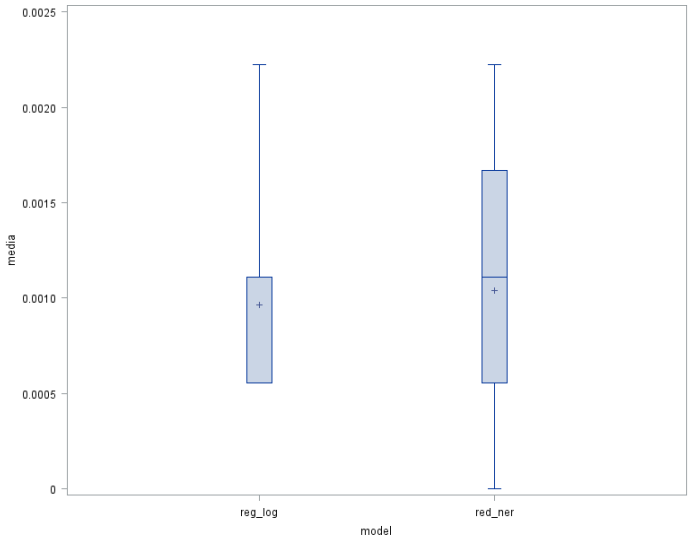
el considerado como mejor modelo de redes neuronales para la clasificación de las sub-imágenes.

Figura 43. Comparación de modelos de redes neuronales simples



Los modelos que contienen una media más baja en la tasa de fallos son el 7, 8 y 11 que son prácticamente iguales, probablemente sean el mismo modelo ya que contienen el mismo número de nodos y el mismo algoritmo de optimización. Por eso, el modelo 7 será el elegido como el mejor modelo y el que se va a comparar con el mejor modelo de regresión logística (figura 44).

Figura 44. Comparación de Regresión Logistica y Redes Neuronales simples



Tal y como indica la Figura 44, el modelo de regresión logística presenta una media para la tasa de aciertos más baja que la red neuronal, aunque la red neural puede llegar a tomar valores más bajos en la tasa de fallos, se va a elegir como mejor modelo la regresión logística debido a que se trata de un modelo mucho más sencillo y presenta una tasa de fallos también muy baja.

Ahora se van a visualizar estadísticos de evaluación del modelo para compararlos con los modelos realizados en Python. Estos estadísticos están realizados sobre los datos que se separaron al principio como conjunto de datos test.

La figura 45 hace referencia a un gráfico de la curva ROC para los datos de test. Por otro lado, la tabla 7 muestra otros estadísticos de evaluación.

Figura 45. Curva de ROC para el conjunto de datos Test (Regresión logística)

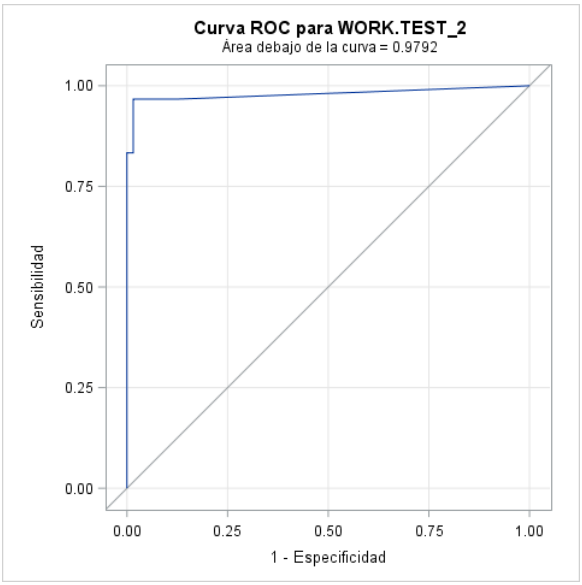


Tabla 7. Estadísticos de bondad de ajuste para el conjunto de datos Test (Regresión Logística)

Estadísticos de ajuste para datos SCORE											
Conjunto de datos	Frecuencia total	Verosimilitud de Log	Tasa de error	AIC	AICC	BIC	SC	R-cuadrado	R-cuadrado máx-reescalada	AUC	Puntuación de Brier
WORK.TEST_2	94	-36.9492	0.0213	81.89831	82.34775	92.07149	92.07149	0.372681	0.521818	0.979167	0.021277

Como se puede observar en la figura 45, el área bajo la curva ROC para los datos de prueba es 0,979167. Esto quiere decir que el modelo de regresión logística con las variables referentes a las medias de R, G y B acierta en casi todos los casos los datos apartados al principio para test.

Para terminar esta parte de clasificación de las sub-imágenes, se va a hacer una comparación entre los mejores modelos elegidos en las redes simples con Python, en las redes convolucionales y la regresión logística.

Tabla 8. Comparación final de los modelos de clasificación

Modelo	AUC Test
Red neuronal con 3 capas ocultas	0,992
ResNet50	0.997
Regresión logística	0,979

En la tabla 8, destaca que el modelo que obtiene un valor para el área de la curva de ROC para la muestra de prueba es la red neuronal ResNet50 sin preentrenar, seguida muy de cerca por la red neuronal con 2 capas ocultas y el valor mas bajo lo toma la regresión logística. Aunque en este caso se cumple que, a mayor complejidad del algoritmos mejor ha clasificado, hay que tener en cuenta que los valores para los tres algoritmos probado son muy parecidos, de hecho, bajo el principio de la parsimonia[15], se ha decidido elegir el modelo de regresión logística como mejor clasificador de sub-imágenes ya que, el valor que obtiene para el área bajo la curva es muy parecido al de los otros algoritmos y se trata de un modelo mucho más sencillo.

5.4. MODELO DE CLASIFICACIÓN PÍXEL BY PIXEL

Comienza la segunda parte del proyecto, encontrar en la foto completa donde se encuentra la mala hierba. Para ello se ha hecho una clasificación por píxel a través de un KNN. Es decir, la idea es crear un modelo que clasifique cada píxel de la imagen como mala hierba o no. Así después, a partir de la probabilidad de que cada píxel contenga o no malezas se representará un mapa de calor que indicará claramente donde se encuentran las malas hierbas en la parcela.

Como se ha comentado en el apartado de metodología, se van a tener en cuenta dos hipótesis para clasificar los píxeles. La primera dicta que no todos los píxeles contenidos en un rectágulo tienen porque ser de la misma clase que dicho rectángulo. La segunda, por el contrario, indica que todos los píxeles de una sub-imagen pertenecen a la misma clase que la sub-imagen. Los próximos apartados explican detalladamente como se ha desarrollado el KNN para cada una de las hipótesis.

5.4.1. HIPÓTESIS 1

Si se tiene en cuenta que no todos los píxeles de una imagen tienen porque ser de la misma clase que la imagen, los datos a utilizar para entrenar el KNN, clasificador por píxeles, son las medias de los canales rojo, verde y azul. Por lo que, simplemente se debe leer el Excel calculado inicialmente en Python y ver qué número de vecinos clasifica obtiene un mayor valor para la sensibilidad. Se va a comparar mediante la sensibilidad

[15] Principio de la parsimonia: Principio metodológico según el cual, en mismas condiciones, la solución más sencilla suele ser la más probable. (Navaja de Ockham)

ya que las zonas donde el modelo indique que hay píxeles con mala hierba, serán los que pasen después a ser tratados por el agricultor. Por lo que, se deben de minimizar los falsos negativos (píxeles donde el modelo indica que no hay mala hierba, pero si la hay). Ya que se estaría descartando una zona que está en mal estado. Aunque también se tendrá en cuenta que la precisión y la exactitud no tomen valores demasiado bajos. Cabe destacar que lo que se está evaluando es como clasifica el KNN la mala hierba del cultivo sano a partir de las medias de los tres canales de color RGB.

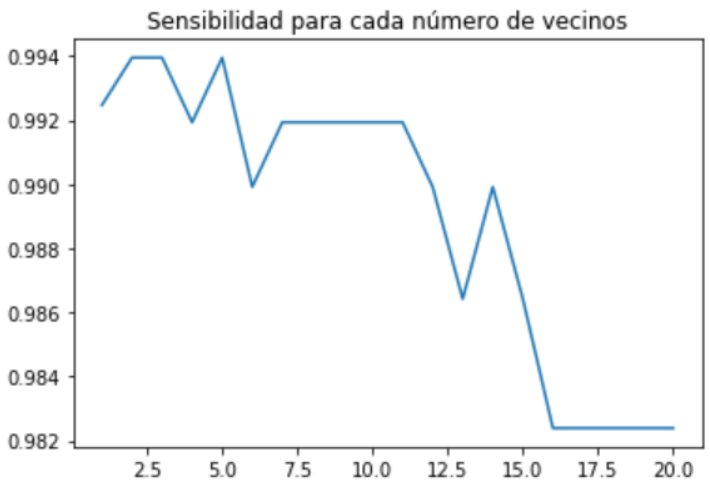
Para elegir un número de vecinos, se ha desarrollado un bucle que recorra el número de vecinos (k) desde 1 hasta 21 y represente la sensibilidad obtenida para cada k. El código es el recopilado en la figura 46

Figura 46. Fragmento de código para determinar el número de vecinos k

```
from sklearn.metrics import recall_score

recall = []
for n in range(1, 21, 1):
  clf = KNeighborsClassifier(n_neighbors=n)
  clf.fit(train[['Rmedia','Gmedia','Bmedia']], train['Clase'])
  clase_pred = clf.predict(test[['Rmedia','Gmedia','Bmedia']])
  k_recall = recall_score(test['Clase'], clase_pred, average='macro')

  recall.append([n, k_recall])
```

El gráfico obtenido es el siguiente (figura 47):

Figura 47. Sensibilidad para cada número de vecinos (hipótesis1)



Como se puede ver, el número de vecinos que obtiene una mejor sensibilidad es 1, 3 o 5. Por lo que, se va a calcular un modelo de clasificación KNN con 3 vecinos. Además de obtener una sensibilidad cercana a 0,995, la tabla 37 compendia otras medidas para la evaluación de modelos.

Tabla 9.Medidas de evaluación orientativas para el KNN por píxel (hipótesis 1)

Medida	valor
Sensibilidad (Recall)	0.9939516129032258
Precisión	0.9897260273972603
Accuracy	0.9923273657289002

Como se puede ver, el valor para las medidas de evaluación es bastante alto, aunque esto es solo orientativo ya que no estamos evaluando los píxeles que se quieren clasificar en realidad. La evaluación de la predicción de esos píxeles se debe hará comparando el mapa de calor obtenido con la imagen original.

El siguiente paso es convertir una imagen en matriz de píxeles utilizando el modelo de color RGB y pasarlo por el modelo construido. Más tarde, se sacará el output del modelo como la probabilidad de que un píxel contenga mala hierba o no y se representará como un mapa de calor.

Figura 48. Fragmento de código para representar el mapa de calor de la imagen completa

```
#Descomposición de las imágenes en matriz RGB

colourImg = Image.open("drive/My Drive/tfm/img_grandes/IMG_5414.JPG")
colourPixels = colourImg.convert("RGB")
colourArray = np.array(colourPixels.getdata()).reshape(colourImg.size + (3,))
indicesArray = np.moveaxis(np.indices(colourImg.size), 0, 2)
allArray = np.dstack((indicesArray, colourArray)).reshape((-1, 5))

df_grand = pd.DataFrame(allArray, columns=["y", "x", "red","green","blue"])

#Otención de la matriz de probilidades

array_p = clf.predict_proba(df_grand[['red','green','blue']])

mat_prob = np.mat(array_p[:,1])
mat_prob = mat_prob.reshape(3000,4000)
print(mat_prob)

#Obtención del mapa de calor

plt.matshow(mat_prob)
plt.show()
```

Por último, se va a representar el mapa de calor de las probabilidades de que cada píxel pertenezca a una clase u otra junto con la imagen original completa.

Figura 49. Mapa de calor (hipótesis 1)

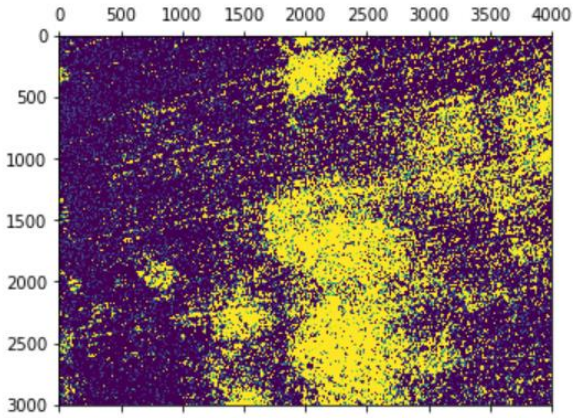


Figura 50. Imagen original completa

Las figuras anteriores muestran que el clasificador por píxeles con 3 vecinos y utilizando las medias de los canales de RGB como patrones funcionan bastante bien para predecir los valores del target. En la figura 49, las zonas más amarillas representan la mala hierba, que se parece bastante a las zonas que realmente tienen mala hierba en la figura 50.

5.4.2. HIPÓTESIS 2

Por último para clasificación píxel por píxel, se va a comentar detalladamente los pasos realizados bajo la hipótesis 2. Recordemos que esta segunda hipótesis afirma que todos los píxeles contenidos en una imagen de una clase pertenecen a esa misma clase, es decir, todos los píxeles habidos en una imagen clasificada como mala hierba contienen mala hierba. Siguiendo este supuesto, se debe entrenar el modelo con los píxeles de las sub-imágenes que el experto clasificó como mala hierba o no considerando que todos los píxeles de una sub-imagen dada pertenecen a esa misma clase. Después, al modelo creado el cual diferencia píxeles con mala hierba de los que no la tienen, le pasamos la imagen completa y devolverá una matriz con de probabilidades, donde cada elemento corresponde a la probabilidad de que ese píxel contenga o no mala hierba.

De la misma manera que bajo la primera conjetura, se va a buscar el modelo que maximice la sensibilidad, pero la manera de obtener los datos para entrenar el modelo cambia totalmente. Primero, se deben desenrollar todos los rectángulos en matriz. La figura 51 contiene el código utilizado para ello.

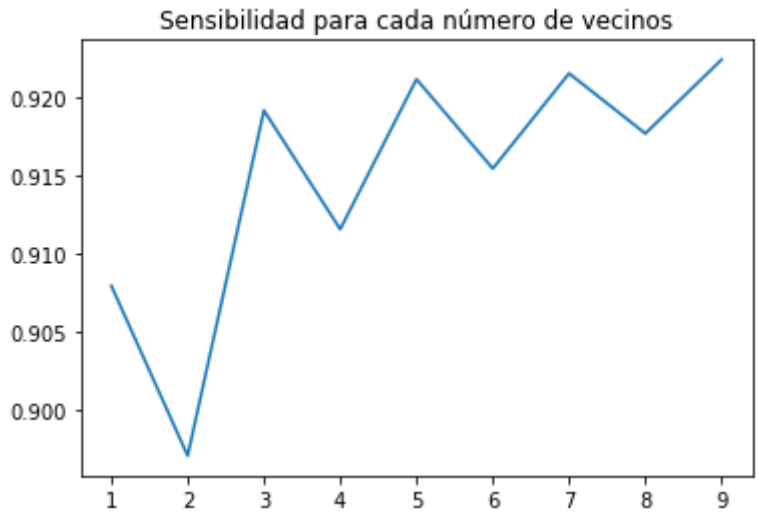
Figura 51. Código para obtener un data frame de píxeles de todas las sub-imágenes

```
df_def = pd.DataFrame([])

for fig in df_img['image_id']:
  fig_str = str(fig)
  filename = "drive/My Drive/tfm/samples/train_data/" + fig_str
  colourImg = Image.open(filename)
  colourPixels = colourImg.convert("RGB")
  colourArray = np.array(colourPixels.getdata()).reshape(colourImg.size + (3,))
  indicesArray = np.moveaxis(np.indices(colourImg.size), 0, 2)
  allArray = np.dstack((indicesArray, colourArray)).reshape((-1, 5))
  df = pd.DataFrame(allArray, columns=["x", "y", "red","green","blue"])
  df["image_id"] = fig_str
  df_def = df_def.append(df)
```

La sintaxis anterior, desenrrolla cada una de las sub-imágenes en píxeles con los canales de color RGB a través de la función “colourImg.convert” de la librería openCV. Después con la librería “pandas” mete los valores del canal rojo, verde y azul en un data frame, teniendo, tantas observaciones como píxeles haya en todas las sub-imágenes en total y como variables se tienen las coordenadas de x e y los valores de los 3 canales de RGB.

Cuando se tiene la tabla deseada, se divide la muestra en entrenamiento y validación y se utiliza el mismo código que en la hipótesis uno para encontrar el número de vecinos que obtenga una mayor sensibilidad. En este caso se ha iterado de 1 a 9 vecinos, ya que, los tiempos de ejecución eran demasiado altos. Este es el gráfico que se obtiene.

Figura 52. Sensibilidad por número de vecinos (hipótesis 2)



Tal y como se puede ver en la figura 52, el número de vecinos que obtiene un valor más alto para la sensibilidad es 9. Seguramente, si se aumenta el número de vecinos, aumentará la sensibilidad, pero la tabla con la que se elabora este gráfico es de altas dimensiones y al ser un proceso iterativo los tiempos de ejecución se disparan. Por otro lado, la tabla 10, resume el valor de varias medidas de evaluación para el KNN con 9 vecinos.

Tabla 10. Medidas de adecuación orientativas KNN por píxeles (hipótesis 2)

Medida	valor
Sensibilidad (Recall)	0.922426076282288
Precisión	0.9548775016714153
Accuracy	0.96831943158354

Los siguientes pasos, consisten en convertir de nuevo la imagen completa en matriz, pasarla por el modelo configurado y representar el mapa de calor de la matriz de las probabilidades de que cada píxel atañe a una clase u otra. El código referente a estas labores es exactamente igual que el desarrollado bajo la primera premisa. Por lo tanto, se va a mostrar la comparación del mapa de calor en este caso con la imagen original completa.

Figura 53. Mapa de calor (Hipótesis 2)

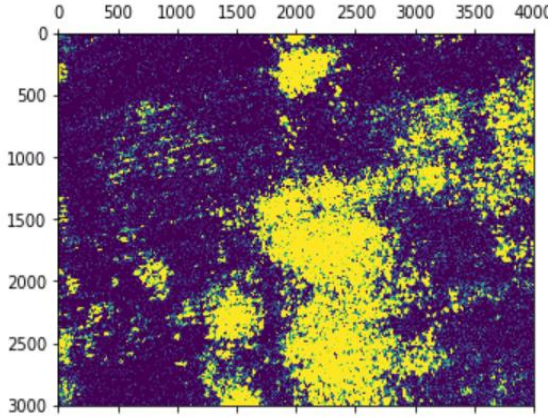


Figura 54. Imagen original

Bajo esta presuposición, los resultados que se obtienen también son bastante buenos, quizá en este caso, el modelo no sea tan sensible y pierda un poco de información o simplemente se obtenga menos ruido en las predicciones.

Por consiguiente, queda claro que ambos resultados son muy buenos, ya que bajo las dos hipótesis el modelo es capaz de detectar perfectamente la mala hierba. El primer modelo (el realizado con las medias), parece ser más sensible a la hora de detectar la mala hierba, es capaz de detectar zonas mucho más pequeñas casi indetectables por un humano a la hora de mirar la imagen. El segundo modelo, se centra más en las zonas donde hay gran cantidad de píxeles con mala hierba, es decir, se fija en las partes donde la mala hierba “abunda”.

6. CONCLUSIONES Y TRABAJO A FUTURO

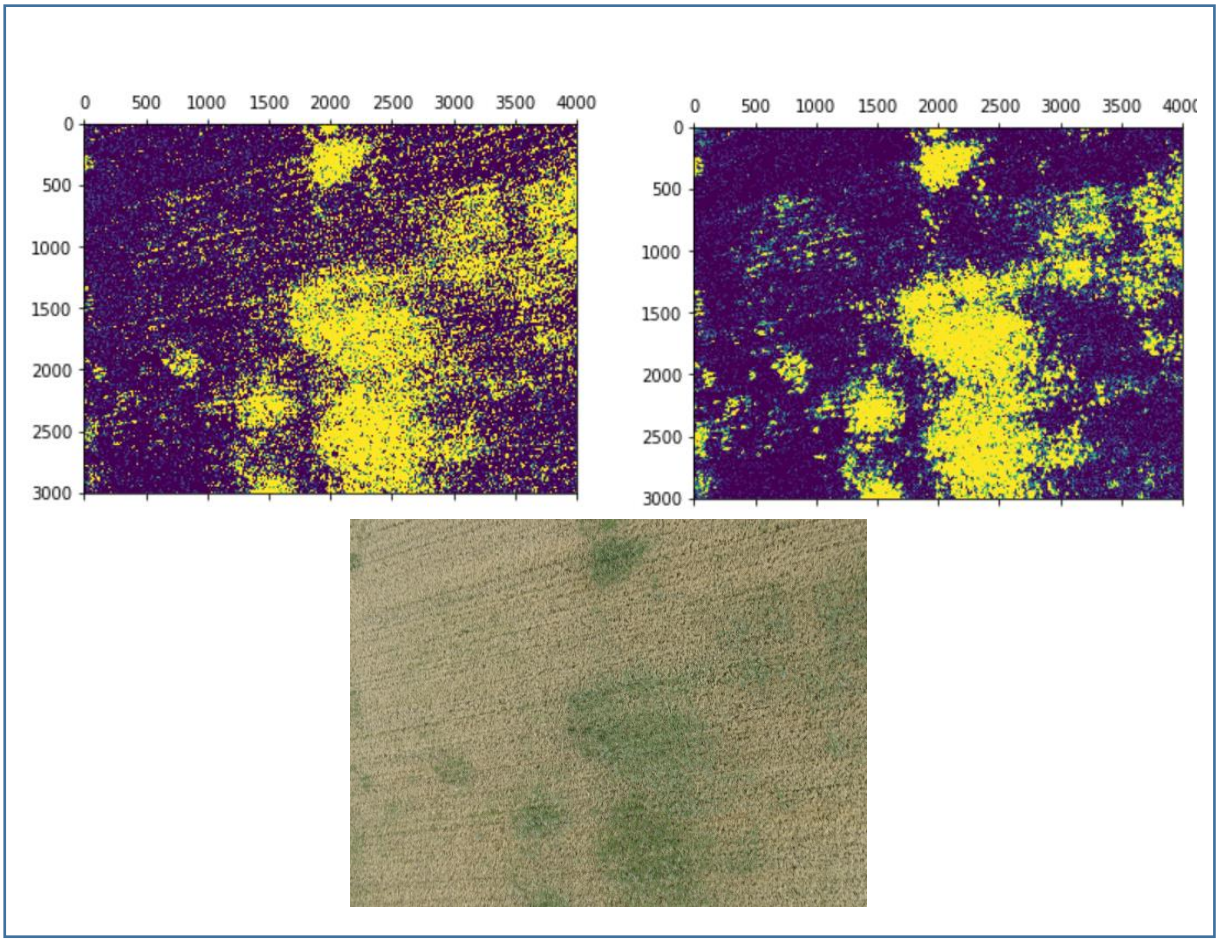
6.1. CONCLUSIONES

Después de probar con diferentes algoritmos de clasificación para encontrar un modelo que mejor clasifique las sub-imágenes, se ha encontrado que se obtienen muy buenos resultados para todos los algoritmos. Por lo que, se ha decidido elegir el más sencillo de todos ellos; la regresión logística con las variables referentes a las medias del color rojo, verde y azul.

Como ya se ha comentado anteriormente durante el apartado de resultados del estudio, se obtienen tan buenos resultados debido a que es muy fácil distinguir una sub-imagen con mala hierba de una que no la tiene, ya que, el mayor detonante a la hora de distinguir ambas clases es el color.

En cuanto a la segunda parte del proyecto, la clasificación píxel by píxel, se tiene que, comparando las dos hipótesis planteadas, el modelo que utiliza las medias de los tres canales RGB como variables es mucho más sensible a la hora de encontrar los patrones de las malas hierbas, lo que hace que obtenga mucho más ruido. Sin embargo, el modelo que utiliza los píxeles en bruto se centra mucho más en las zonas donde la mala hierba abunda, es decir, en zonas donde hay un conjunto de píxeles considerable con malezas herbáceas. Quizás, a la hora de tomar medidas es más útil el segundo modelo ya que si se van a usar herbicidas o se va a segar en cierta zona, será en un sitio donde haya una pequeña concentración de malezas en la tierra. Pero si se quiere saber cuál es el estado de salud general del terreno, puede ser más útil el primer modelo. La figura 42 es una comparación entre los resultados de los dos supuestos y la imagen original de uno de los campos de cultivo:

Figura 55. Comparación entre los resultados de los dos modelos de clasificación por píxel y la imagen original



Si se tiene en cuenta el principio de la parsimonia, el primer modelo necesita mucho menos tiempo de ejecución que el segundo, ya que el número de observaciones con las que se realiza el modelo es mucho menor que si se utilizan los píxeles en bruto para entrenar el KNN. Debido a que es un modelo más sencillo y los resultados obtenidos bajo las dos conjeturas son parecidos, se ha decidido que el modelo más óptimo de clasificación píxel by píxel es el realizado con las medias de los colores rojo, verde y azul.

Teniendo en cuenta los objetivos secundarios del estudio, en este caso, ha sido preferible utilizar modelos sencillos con un set de datos pequeño y una variable respuesta de solamente dos categorías, a utilizar arquitecturas complejas o de aprendizaje profundo. Aunque en ambas se hayan obtenido buenos resultados, la regresión logística requiere mucho menos tiempo computacional y además es mucho más sencilla de comprender, llevar a cabo y sus resultados son más fáciles de interpretar.

6.2. TRABAJO A FUTURO

Mirando al futuro, para mejorar este estudio o encontrar nuevas soluciones ante el problema de clasificación tanto de las imágenes como de los píxeles, se podría probar a aumentar la muestra o probar nuevos algoritmos de clasificación como, por ejemplo, algoritmos basados en árboles.

Otro enfoque interesante para el estudio sería utilizar un canal de color distinto al RGB y ver si mejoran los resultados utilizando otro canal, como, por ejemplo, el espacio de color CMYK (Cian, Magenta, Amarillo y Negro).

7. BIBLIOGRAFÍA

De la Escalera, A. (2001). Preprocesamiento de imágenes. *Visión Por Computador, Fundamentos y Métodos*, 112–154.

De, M., & Datos, M. De. (2019). *Técnicas y Metodología de la Minería de Datos (SEMMA) Máster en Minería de Datos e Inteligencia de Negocio*.

Delgado-Gutiérrez, M. J., Herrera-Guillén, D. F., Medina-Barragán, L. M., & Corredor–Gómez, J. P. (2017). Implementación de un sistema de procesamiento de imágenes integrado con Raspberry PI 2B para reconocimiento y recolección de fresas maduras. *Revista Politécnica*, *13*(25), 75–85. https://doi.org/10.33571/rpolitec.v13n25a6

Miguel, J. (2015). *Sistema de visión para agricultura: Identificación en tiempo real de lineas de cultivo y malas hierbas en campos de maíz*.

Valero Ubierna, C. (2001). Agricultura de precisión: conceptos y situación actual. *Vida Rural*, *136*(136), 58–62. Retrieved from http://oa.upm.es/6291/1/Valero_36.pdf%0Afile:///D:/Àlex/Feina/Bibliografia/Articles i estudis/AP/Agricultura de precisión. conceptos y situación actual - Valero (VR136 2001).pdf%0Ahttp://oa.upm.es/6291/

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

https://es.wikipedia.org/wiki/Agricultura_de_precisi%C3%B3n

https://www.tecnologiahorticola.com/la-agricultura-de-precision-como-se-aplica/

https://www.apd.es/que-es-deep-learning/

https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales

http://www.diegocalvo.es/red-neuronal-convolucional/

https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

https://cleverdata.io/que-es-machine-learning-big-data/

https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada

http://www.radulucaciu.ro/repairing-an-image-in-python-using-machine-learning-using-knn-filters/103/

https://scikit-learn.org/stable/modules/neighbors.html

https://www.researchgate.net/publication/28244234_Logit_Model_como_modelo_de_eleccion_discreta_origen_y_evolucion

https://es.wikipedia.org/wiki/Logit

http://avellano.fis.usal.es/~lalonso/RNA/index.htm

https://disi.unal.edu.co/~lctorress/RedNeu/LiRna008.pdf

https://es.wikipedia.org/wiki/Propagaci%C3%B3n_hacia_atr%C3%A1s

https://online.datademy.es/redes-neuronales-en-sas/

http://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm

https://www.monografias.com/trabajos-pdf/entrenamiento-redes-neuronales-algoritmos-evolutivos/entrenamiento-redes-neuronales-algoritmos-evolutivos2.shtml

https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/

https://www.analiticaweb.es/algoritmo-knn-modelado-datos/

https://la.nvidia.com/object/what-is-gpu-computing-la.html

https://es.wikipedia.org/wiki/Unidad_de_procesamiento_gráfico

https://www.xdrones.es/drones-para-agricultura-de-precision/

https://www.ainia.es/tecnoalimentalia/tecnologia/drones-agricultura-de-precision-e-industria-alimentaria-nuevas-tendencias/

https://optimization.mccormick.northwestern.edu/index.php/Trust-region_methods

http://www.grap.udl.cat/es/presentacion/ap.html

https://www.ainia.es/tecnoalimentalia/tecnologia/drones-agricultura-de-precision-e-industria-alimentaria-nuevas-tendencias/

https://programarfacil.com/blog/vision-artificial/aprender-vision-artificial/

8. ANEXO

8.1. ANEXO DE CÓDIO SAS

```
libname tfm_data 'D:\TFM\para_sas\libreria';

data tfm_data.datos_completos;
     set data_complet;
     id = _N_;
run;


proc sql;
create table tfm_data.test as
select a.id, a.clase, a.Rmedia, a.Gmedia, a.bmedia
from tfm_data.datos_completos as a inner join tfm_data.id_test
as b
on a.id = b.id_imagen;
quit;


proc sql;
create table tfm_data.train as
select a.id, a.clase, a.Rmedia, a.Gmedia, a.bmedia
from tfm_data.datos_completos as a inner join tfm_data.id_train
as b
on a.id = b.id_imagen;
quit;

proc means data = tfm_data.train max min mean std mode q1 median
q3;
     var Rmedia gmedia bmedia;
run;


proc univariate data = tfm_data.train;
     var Rmedia gmedia bmedia; hist Rmedia gmedia bmedia /
kernel normal;
     inset mean std;
run;

libname tfm_data 'D:\TFM\para_sas\libreria';

proc means data = tfm_data.train max min mean std mode q1 median
q3;
     var Rmedia gmedia bmedia;
     where clase = 0;
run;

proc means data = tfm_data.train max min mean std mode q1 median
q3;
     var Rmedia gmedia bmedia;
     where clase = 1;
run;
```

```
proc univariate data = tfm_data.train;
      var Rmedia gmedia bmedia; hist Rmedia gmedia bmedia /
kernel normal;
      inset mean std;
      where clase = 0;
run;



proc univariate data = tfm_data.train;
      var Rmedia gmedia bmedia; hist Rmedia gmedia bmedia /
kernel normal;
      inset mean std;
      where clase = 1;
run;


/**********************REGRESIÓN
LOGÍSTICA***************************/


/*
*************************************************************
*************
MACRO INTERACTTODOLOG
*************************************************************
*************

%macro
interacttodolog(archivo=,vardep=,listclass=,listconti=,interac=1
,directorio=c:\Windows\Temp);

La macro INTERACTTODOLOG calcula un listado de interacciones
entre:

* variables categóricas hasta orden 2
* variables continuas y categóricas (hasta orden 2)

Y además presenta un listado de las variables e interacciones
por orden de pvalor
consideradas individualmente en un proc LOGISTIC.

archivo=
depen=variable dependiente
listclass=lista de variables de clase
listconti=lista de variables continuas
interac= 1 si se quieren interacciones(puede tardar mucho
dependiendo de la complejidad)
           (interac=0 si no se quieren interacciones)
directorio= poner el directorio para archivos temporales basura

********
SALIDA
```

```
********

EL ARCHIVO CONSTRUIDO POR LA MACRO SE LLAMA UNION. CONTIENE LOS
EFECTOS ORDENADOS
POR ORDEN ASCENDENTE DE AIC (CUANTO MÁS PEQUEÑO MEJOR).
TAMBIÉN SE PUEDE REORDENAR POR ORDEN DESCENDENTE DEL VALOR DEL
ESTADÍSTICO F (CUANTO MAYOR MEJOR)
O POR PVALOR DEL CONTRASTE (MÁS PEQUEÑO MEJOR).

UNA VEZ EJECUTADA LA MACRO SE PUEDE OBTENER UN LISTADO DE LOS
EFECTOS
EN EL LOG POR ORDEN DE MEJOR AIC A PEOR, CON:

data _null_;set union;put variable @@;run;

************************************************************
**************

NOTAS Y TRUCOS PARA SU APROVECHAMIENTO

1) ANTE ARCHIVOS CON MUCHAS VARIABLES CATEGÓRICAS SE PUEDE
EJECUTAR POR PARTES,
POR EJEMPLO:
a)SOLO CONTINUAS
b)SOLO CATEGÓRICAS, CON O SIN INTERACCIONES
c)ELEGIR LAS K MEJORES CATEGÓRICAS y/o CONTINUAS Y VOLVER A
EJECUTAR CON INTERACCIONES

2) EN GENERAL ANTE MUCHAS VARIABLES CATEGÓRICAS Y VARIABLES
CATEGÓRICAS
CON MUCHAS CATEGORÍAS ES MEJOR REFINAR LOS DATOS UTILIZANDO LA
MACRO
NOMBRESMOD ANTES. PERO PARA ELLO SE NECESITA A MENUDO UNA
PRESELECCIÓN
CON EL APARTADO b ANTERIOR

3) EL ORDEN OBTENIDO NO ES DETERMINANTE PARA EL MODELO (SERÁ
NECESARIO UTILIZAR
TÉCNICAS TIPO STEPWISE TAMBIÉN) PERO SÍ PARA UNA PRESELECCIÓN Y
RECHAZO DE EFECTOS
QUE NO SIRVEN.

4) PUEDE PROBARSE ANTES CON AGRUPARCATEGORIAS

*/

%macro
interacttodolog(archivo=,vardep=,listclass=,listconti=,interac=1
,directorio=c:);
proc printto print="&directorio\kaka.txt";run;
data _null_;
file "&directorio\inteconti.txt";
put ' ';
file "&directorio\intecategor.txt";
put ' ';
run;
```

```
data _null_;
length clase conti con cruce1 $ 32000 cruce2 $ 32000;
clase="&listclass";
conti="&listconti";
  ncate= 1;
  do while (scan(clase, ncate) ^= '');
    ncate+1;
  end;
  ncate+(-1);
  put ncate=;
  nconti= 1;
  do while (scan(conti, nconti) ^= '');
    nconti+1;
  end;
  nconti+(-1);
  put nconti=;

call symput('ncate',left(ncate));
call symput('nconti',left(nconti));

%if &interac=1 %then %do;
cruce2=' ';
do i=1 to ncate;
      do j=1 to nconti;
      ca=scan(clase,i);
            con=scan(conti,j);
            cruce1=cats(ca,'*',con);
            file "&directorio\inteconti.txt" mod;
            put cruce1;
      end;
end;

cruce2=' ';
do i=1 to ncate-1;
      do j=i+1 to ncate;
      ca=scan(clase,i);
            con=scan(clase,j);
            if i ne j then cruce1=cats(ca,'*',con);else cruce1='
';
            file "&directorio\intecategor.txt" mod;
            put cruce1;
      end;
end;
run;
%end;
data union;run;

/* variables de clase solitas */
%if &listclass ne %then %do i=1 %to &ncate;
data _null_;cosa="&listclass";va=scanq(cosa,&i);
call symput ('vari',va);
run;

ods output Globaltests=global association=asoc;
ods output Fitstatistics=Fitstatistics;
```

```
proc logistic data=&archivo;
class &vari;
model &vardep=&vari;
run;

data c (keep=variable percentconcord AIC Chisq ProbChisq);length
variable $ 1000;
set global;set asoc;set fitstatistics;
AIC=InterceptandCovariates;percentconcord=cvalue1;
variable="&vari";
if _n_=1 then output;
run;

data union;set union c;run;

%end;

/* interacciones de variables de clase */

%if &interac=1 %then %do;

%if &ncate>1 %then %do;

data pr234;
length vari $ 1000;
infile "&directorio\intecategor.txt";
input vari;
run;
data _null_;set pr234 nobs=nume;ko=nume;
call symput('nintecat',left(ko));stop;
run;

%if &listclass ne %then %do i=1 %to &nintecat;
data _null_;ko=&i;
set pr234 point=ko;
var1=scan(vari,1);
var2=scan(vari,2);
lista1=compbl(var1||'  '||var2);
call symput('lista1',left(lista1));
call symput('vari',left(vari));
stop;
run;

ods output Globaltests=global association=asoc;
ods output Fitstatistics=Fitstatistics;
proc logistic data=&archivo;
class &lista1;
model &vardep=&vari;
run;

data c (keep=variable percentconcord AIC Chisq ProbChisq);length
variable $ 1000;
set global;set asoc;set fitstatistics;
AIC=InterceptandCovariates;percentconcord=cvalue1;
variable="&vari";
if _n_=1 then output;
```

```
run;

data union;set union c;run;

%end;
data _null_;if _n_=1 then put 'LISTA CLASE E INTERACCIONES';set
union;put variable @@;run;
%end;

%end;

/* variables continuas solitas */
%if &listconti ne %then %do i=1 %to &nconti;
data _null_;cosa="&listconti";va=scanq(cosa,&i);
call symput ('vari',va);
run;

ods output Globaltests=global association=asoc;
ods output Fitstatistics=Fitstatistics;
proc logistic data=&archivo;
model &vardep=&vari;
run;

data c (keep=variable percentconcord AIC Chisq ProbChisq);length
variable $ 1000;
set global;set asoc;set fitstatistics;
AIC=InterceptandCovariates;percentconcord=cvalue1;
variable="&vari";
if _n_=1 then output;
run;

data union;set union c;run;

%end;

/* interacciones de variables de clase con variables continuas
*/
%if &interac=1 %then %do;
data pr235;
length vari $ 1000;
infile "&directorio\inteconti.txt";
input vari;
run;

data _null_;set pr235 nobs=nume;ko=nume;
call symput('ninteconti',left(ko));stop;
run;

%if (&listclass ne) and (&listconti ne) %then %do i=1 %to
&ninteconti;
data _null_;ko=&i;
set pr235 point=ko;
var1=scan(vari,1);
var2=scan(vari,2);
call symput('lista1con',left(var1));
call symput('varicon',left(vari));
```

```
stop;
run;

ods output Globaltests=global association=asoc;
ods output Fitstatistics=Fitstatistics;
proc logistic data=&archivo;
class &lista1con;
model &vardep=&varicon;
run;

data c (keep=variable percentconcord AIC Chisq ProbChisq);length
variable $ 1000;
set global;set asoc;set fitstatistics;
AIC=InterceptandCovariates;percentconcord=cvalue1;
variable="&varicon";
if _n_=1 then output;
run;

data union;set union c;run;

%end;
%end;
proc printto;run;
data union;set union;if _n_=1 then delete;run;
proc sort data=union;by AIC;
proc print data=union;run;
data _null_;set union;put variable @@;run;
%mend;


%interacttodolog(archivo=tfm_data.train,vardep=clase,listclass=m
ediar mediag mediab,
listconti=,interac=1,directorio=D:\MASTER\2cuatri\redes
neuronales\PRACTICA2\kk);


/* VALIDACIÓN CRUZADA LOGÍSTICA PARA VARIABLES DEPENDIENTES
BINARIAS

**********************************************************
****************
                                          PARÁMETROS
**********************************************************
****************

BÁSICOS

archivo=        archivo de datos
vardepen=       variable dependiente binaria
categor=        lista de variables independientes categóricas
conti=               lista de variables independientes
continuas Y TODAS LAS INTERACCIONES
ngrupos=        número de grupos validación cruzada
sinicio=        semilla inicial para repetición
sfinal=              semilla final para repetición
```

```
objetivo=
     tasafallos,sensi,especif,porcenVN,porcenFN,porcenVP,porcenF
P,precision,tasaciertos


El archivo final se llama final. La variable media es la media
del obejtivo en todas las pruebas de validación cruzada
(habitualmente tasa de fallos).

*/

%macro
cruzadalogistica(archivo=,vardepen=,conti=,categor=,ngrupos=,sin
icio=,sfinal=,objetivo=);
title ' ';
data final;run;
/* Bucle semillas */
%do semilla=&sinicio %to &sfinal;
     data dos;set &archivo;u=ranuni(&semilla);
     proc sort data=dos;by u;run;
     data dos (drop=nume);
     retain grupo 1;
     set dos nobs=nume;
     if _n_>grupo*nume/&ngrupos then grupo=grupo+1;
     run;
     data fantasma;run;
     %do exclu=1 %to &ngrupos;
          data tres;set dos;if grupo ne &exclu then
vardep=&vardepen;
          proc logistic data=tres noprint;/*<<<<<******SE PUEDE
QUITAR EL NOPRINT */
          %if (&categor ne) %then %do;class &categor;model
vardep=&conti &categor ;%end;
          %else %do;model vardep=&conti;%end;
          output out=sal p=predi;run;
          data sal2;set sal;pro=1-predi;if pro>0.5 then
pre11=1; else pre11=0;
          if grupo=&exclu then output;run;
          proc freq data=sal2;tables
pre11*&vardepen/out=sal3;run;
          data estadisticos (drop=count percent pre11
&vardepen);
          retain vp vn fp fn suma 0;
          set sal3 nobs=nume;
          suma=suma+count;
          if pre11=0 and &vardepen=0 then vn=count;
          if pre11=0 and &vardepen=1 then fn=count;
          if pre11=1 and &vardepen=0 then fp=count;
          if pre11=1 and &vardepen=1 then vp=count;
          if _n_=nume then do;
          porcenVN=vn/suma;
          porcenFN=FN/suma;
          porcenVP=VP/suma;
          porcenFP=FP/suma;
          sensi=vp/(vp+fn);
          especif=vn/(vn+fp);
```

```
            tasafallos=1-(vp+vn)/suma;
            tasaciertos=1-tasafallos;
            precision=vp/(vp+fp);
            F_M=2*Sensi*Precision/(Sensi+Precision);
            output;
            end;
            run;

            data fantasma;set fantasma estadisticos;run;
      %end;
      proc means data=fantasma sum noprint;var &objetivo;
      output out=sumaresi sum=suma mean=media;
      run;
      data sumaresi;set sumaresi;semilla=&semilla;
      data final (keep=suma media semilla);set final sumaresi;if
suma=. then delete;run;
%end;
proc print data=final;run;
%mend;


%cruzadalogistica(archivo=tfm_data.train,
vardepen=Clase,
conti = rmedia gmedia bmedia,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,objetivo=tasafallos sensi
especif porcenVN porcenFN porcenVP porcenFP precision);


data modelo1;
      set final;
      modelo = 1;
run;

/*tasafallos,sensi,especif,porcenVN,porcenFN,porcenVP,porcenFP,p
recision,tasaciertos*/


%cruzadalogistica(archivo=tfm_data.train,
vardepen=Clase,
conti = rmedia gmedia bmedia rmedia*gmedia rmedia*bmedia
gmedia*bmedia,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,objetivo=tasafallos sensi
especif porcenVN porcenFN porcenVP porcenFP precision);

data modelo2;
      set final;
      modelo = 2;
run;

data union;
      set modelo1 modelo2;
run;

proc boxplot data=union;
```

```
     plot media*modelo;
run;

/************************REDES
NEURONALES************************/

/*macro fuente del apartado de redes*/

%macro
neuralbinariabasica(archivo=,listconti=,listclass=,vardep=,nodos
=,corte=,semilla=,porcen=,algo=levmar);
title '';
data archivobase;set &archivo nobs=nume;ene=int(&porcen*nume);
call symput('ene',left(ene));
run;

proc sort data=archivobase;by &vardep;run;

proc surveyselect data=archivobase out=muestra outall N=&ene
seed=&semilla;
/*si se quiere estratificacion en el muestreo quitar los
comentarios en strata*/
/* strata &vardep /alloc=proportional;*/run;
data train valida;set muestra;if selected=1 then output
train;else output valida;run;

PROC DMDB DATA=train dmdbcat=cataprueba;
target &vardep;
var &listconti;
class &listclass &vardep;
run;

%if &listclass ne %then %do;
proc neural data=train dmdbcat=cataprueba;
input  &listconti;
input &listclass /level=nominal;
target  &vardep /level=nominal;
hidden &nodos;
prelim 5;
train tech=&algo;
score data=valida out=salpredi outfit=salfit ;
run;
%end;

%else %do;
proc neural data=train dmdbcat=cataprueba;
input  &listconti;
target  &vardep /level=nominal;
hidden &nodos;
prelim 5;
train tech=&algo;
score data=valida out=salpredi outfit=salfit ;
run;
%end;
```

```
data salpredi;set salpredi;if p_&vardep.1>&corte/100 then
predi1=1;else predi1=0;run;
proc freq data=salpredi;tables predi1*&vardep/out=sal1;run;

/* Cálculo de estadísticos */

data estadisticos (drop=count percent predi1 &vardep);
retain vp vn fp fn suma 0;
set sal1 nobs=nume;
suma=suma+count;
if predi1=0 and &vardep=0 then vn=count;
if predi1=0 and &vardep=1 then fn=count;
if predi1=1 and &vardep=0 then fp=count;
if predi1=1 and &vardep=1 then vp=count;
if _n_=nume then do;
if vn=. then vn=0;if fn=. then fn=0;if vp=. then vp=0;if fp=.
then fp=0;
porcenVN=vn/suma;
porcenFN=FN/suma;
porcenVP=VP/suma;
porcenFP=FP/suma;
sensi=vp/(vp+fn);
especif=vn/(vn+fp);
tasafallos=1-(vp+vn)/suma;
tasaciertos=1-tasafallos;
precision=vp/(vp+fp);
if vp=0 then precision=0;
if vp=0 then sensi=0;
if vn=0 then especif=0;
F_M=2*Sensi*Precision/(Sensi+Precision);
output;
end;
run;
proc print data=estadisticos;run;

%mend;

%neuralbinariabasica(archivo=tfm_data.train,
listconti=rmedia gmedia bmedia,
listclass=,
vardep=clase,nodos=2,semilla=45,corte=50,porcen=0.8,algo=levmar)
;

/*para elegir número de nodos*/


%numeronodos(inicionodos=1,finalnodos=3,increnodos=1);


%macro
variar(seminicio=,semifin=,inicionodos=,finalnodos=,increnodos=)
;
title '';
data union;run;
%do semilla=&seminicio %to &semifin;
%do nodos=&inicionodos %to &finalnodos %by &increnodos;
```

```
   %neuralbinariabasica(archivo=tfm_data.train,
   listconti=rmedia gmedia bmedia,

listclass=,vardep=Clase,nodos=&nodos,corte=50,semilla=12345,porc
en=0.80);
   data estadisticos;set
estadisticos;nodos=&nodos;semilla=&semilla;run;
   data union;set union estadisticos;run;
%end;
%end;
proc sort data=union;by nodos;run;
proc boxplot data=union;plot (porcenVN porcenFN porcenVP
porcenFP
sensi especif tasafallos tasaciertos precision F_M)*nodos;run;
%mend;

%variar(seminicio=12345,semifin=12362,inicionodos=1,finalnodos=3
0,increnodos=2);


/*macro generica*/



%macro
cruzadabinarianeural(archivo=,vardepen=,conti=,categor=,ngrupos=
,sinicio=,sfinal=,nodos=,algo=,objetivo=,early=300,
acti=tanh,basura='D:');
title ' ';
data final;run;
proc printto print=&basura;

/* Bucle semillas */
%do semilla=&sinicio %to &sfinal;
     data dos;set &archivo;u=ranuni(&semilla);
     proc sort data=dos;by u;run;
     data dos (drop=nume);
     retain grupo 1;
     set dos nobs=nume;
     if _n_>grupo*nume/&ngrupos then grupo=grupo+1;
     run;
     data fantasma;run;
     %do exclu=1 %to &ngrupos;

     data trestr tresval;
          set dos;if grupo ne &exclu then output trestr;else
output tresval;
          PROC DMDB DATA=trestr dmdbcat=catatres;
          target &vardepen;
          var &conti;
          class &vardepen;
          %if &categor ne %then %do;class &categor
&vardepen;%end;
          run;
          proc neural data=trestr dmdbcat=catatres random=789 ;
          input &conti;
```

```
           %if &categor ne %then %do;input &categor
/level=nominal;%end;
           target &vardepen /level=nominal;
           hidden &nodos /acti=&acti;/*<<<<<******PARA DATOS
LINEALES ACT=LIN (funciónde activación lineal)
           NORMALMENTE PARA DATOS NO LINEALES MEJOR ACT=TANH */
           /* A PARTIR DE AQUI SON ESPECIFICACIONES DE LA RED,
SE PUEDEN CAMBIAR O AÑADIR COMO PARÁMETROS */

           /*nloptions maxiter=500*/;
           netoptions randist=normal ranscale=0.15 random=15459;
           /* Si se desea hacer early stopping se pone prelim 0
y se marca como comentario
           la línea prelim 15...*/
           /*prelim 0 */
           prelim 15 preiter=10 pretech=&algo;
           train maxiter=&early outest=mlpest technique=&algo;
           score data=tresval role=valid out=sal ;
           run;
           data sal2;set sal;pro=1-%str(p_&vardepen)0;if pro>0.5
then pre11=1; else pre11=0;run;
                      proc freq data=sal2;tables
pre11*&vardepen/out=sal3;run;

     data estadisticos (drop=count percent pre11 &vardepen);
           retain vp vn fp fn suma 0;
           set sal3 nobs=nume;
           suma=suma+count;
           if pre11=0 and &vardepen=0 then vn=count;
           if pre11=0 and &vardepen=1 then fn=count;
           if pre11=1 and &vardepen=0 then fp=count;
           if pre11=1 and &vardepen=1 then vp=count;
           if _n_=nume then do;
           porcenVN=vn/suma;
           porcenFN=FN/suma;
           porcenVP=VP/suma;
           porcenFP=FP/suma;
           sensi=vp/(vp+fn);
           especif=vn/(vn+fp);
           tasafallos=1-(vp+vn)/suma;
           tasaciertos=1-tasafallos;
           precision=vp/(vp+fp);
           F_M=2*Sensi*Precision/(Sensi+Precision);
           output;
           end;
           run;
     data fantasma;set fantasma estadisticos;run;
     %end;
     proc means data=fantasma sum noprint;var &objetivo;
     output out=sumaresi sum=suma mean=media;
     run;
     data sumaresi;set sumaresi;semilla=&semilla;
     data final (keep=suma media semilla);set final sumaresi;if
suma=. then delete;run;
%end;
proc printto ;
```

```
proc print data=final;run;
%mend;


/*algoritmo de optimización 1 nodo*/

%macro algovalcruza;
%let lista='BPROP LEVMAR QUANEW TRUREG';
%let nume=4; %do  i=1 %to &nume;
data _null_;
meto=scanq(&lista,&i);
call symput('meto',left(meto));
run;
%cruzadabinarianeural (archivo=tfm_data.train,
vardepen=clase,conti=rmedia gmedia bmedia,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=1,algo=&meto,objetivo
=tasafallos,early=300,
acti=tanh,basura=D:);

data final&i;set final;modelo="&meto";put modelo=;run; %end;
data union;set %do i=1 %to &nume; final&i %end;
%mend;

%algovalcruza;

data union;
set final1 final2 final3 final4;
run;


proc boxplot data=union;
     plot media*modelo;
run;


/*algoritmo de optimización 9 nodos*/

%macro algovalcruza;
%let lista='BPROP LEVMAR QUANEW TRUREG';
%let nume=4; %do  i=1 %to &nume;
data _null_;
meto=scanq(&lista,&i);
call symput('meto',left(meto));
run;
%cruzadabinarianeural (archivo=tfm_data.train,
vardepen=clase,conti=rmedia gmedia bmedia,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=9,algo=&meto,objetivo
=tasafallos,early=300,
acti=tanh,basura=D:);

data final&i;set final;modelo="&meto";put modelo=;run; %end;
data union;set %do i=1 %to &nume; final&i %end;
%mend;
```

```
%algovalcruza;


data union;
set final1 final2 final3 final4;
run;


proc boxplot data=union;
     plot media*modelo;
run;


/*funcion de activacion 1 nodo*/

%macro activalcruza;
%let lista='TANH LOG ARC SIN SOF LOGISTIC';
%let nume=6; %do  i=1 %to &nume;
data _null_;
activa=scanq(&lista,&i);
call symput('activa',left(activa));
run;
%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=1,
algo=BPROP MOM=0.7 LEARN=0.3,acti=&ACTIVA,objetivo=tasafallos);
data final&i;
set final;
modelo="&activa";
put modelo=;
run;
%end;
data union;
set %do i=1 %to &nume;
final&i %end;
%mend;


%activalcruza;

data union;
set final1 final2 final3 final4 final5 final6;
run;


proc boxplot data=union;
     plot media*modelo;
run;



/*función de actvación 9 nodos
*/

%macro activalcruza;
```

```
%let lista='TANH LOG ARC SIN SOF LOGISTIC';
%let nume=6; %do  i=1 %to &nume;
data _null_;
activa=scanq(&lista,&i);
call symput('activa',left(activa));
run;
%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=1,
algo=TRUREG,acti=&ACTIVA,objetivo=tasafallos);
data final&i;
set final;
modelo="&activa";
put modelo=;
run;
%end;
data union;
set %do i=1 %to &nume;
final&i %end;
%mend;


%activalcruza;


data union;
set final1 final2 final3 final4 final5 final6;
run;


proc boxplot data=union;
     plot media*modelo;
run;


%macro activalcruza;
%let lista='TANH LOG ARC SIN SOF LOGISTIC';
%let nume=6; %do  i=1 %to &nume;
data _null_;
activa=scanq(&lista,&i);
call symput('activa',left(activa));
run;
%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=9,
algo=QUANEW,acti=&ACTIVA,objetivo=tasafallos);
data final&i;
set final;
modelo="&activa";
put modelo=;
run;
%end;
data union;
set %do i=1 %to &nume;
```

```
final&i %end;
%mend;

%activalcruza;


data union;
set final1 final2 final3 final4 final5 final6;
run;


proc boxplot data=union;
     plot media*modelo;
run;


%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia ,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=1,
algo=BPROP MOM=0.7 LEARN=0.3,acti=tanh,objetivo=tasafallos);

data tfm_data.final1;
set final;
modelo=1;
run;

%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia ,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=1,
algo=trureg,acti=tanh,objetivo=tasafallos);

data tfm_data.final2;
set final;
modelo=2;
run;

%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia ,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=1,
algo=trureg,acti=log,objetivo=tasafallos);

data tfm_data.final3;
set final;
modelo=3;
run;


%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia ,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=1,
algo=trureg,acti=arc,objetivo=tasafallos);
```

```
data tfm_data.final4;
set final;
modelo=4;
run;


%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia ,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=1,
algo=trureg,acti=sin,objetivo=tasafallos);

data tfm_data.final5;
set final;
modelo=5;
run;


%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia ,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=1,
algo=trureg,acti=logistic,objetivo=tasafallos);

data tfm_data.final6;
set final;
modelo=6;
run;


%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia ,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=9,
algo=QUANEW,acti=tanh,objetivo=tasafallos);

data tfm_data.final7;
set final;
modelo=7;
run;


%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia ,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=9,
algo=QUANEW,acti=log,objetivo=tasafallos);

data tfm_data.final8;
set final;
modelo=8;
run;


%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia ,
```

```
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=9,
algo=QUANEW,acti=arc,objetivo=tasafallos);

data tfm_data.final9;
set final;
modelo=9;
run;


%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia ,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=9,
algo=QUANEW,acti=sin,objetivo=tasafallos);

data tfm_data.final10;
set final;
modelo=10;
run;


%cruzadabinarianeural(archivo=tfm_data.train,vardepen=clase,
conti=rmedia gmedia bmedia ,
categor=,
ngrupos=5,sinicio=12345,sfinal=12367,nodos=9,
algo=QUANEW,acti=logistic,objetivo=tasafallos);

data tfm_data.final11;
set final;
modelo=11;
run;


data union;
set tfm_data.final1 tfm_data.final2 tfm_data.final3
tfm_data.final4 tfm_data.final5 tfm_data.final6 tfm_data.final7
tfm_data.final8
     tfm_data.final9 tfm_data.final10 tfm_data.final11;
run;


proc boxplot data=union;
     plot media*modelo;
run;



data modelo1_reg;
set tfm_data.modelo1_reg;
model = 'reg_log';
run;


data modelo7_ref;
```

```
set tfm_data.final7 ;
model = 'red_nerual';
run;



data union_reg_red;
set modelo1_reg modelo7_ref ;
run;


proc boxplot data=union_reg_red;
      plot media*model;
run;




data test;
set tfm_data.test;
run;


ods graphics on;
proc logistic data=tfm_data.train;
           class Clase;
         model Clase = Rmedia Gmedia Bmedia / outroc=troc;
         score data=test out=valpred fitstat outroc=vroc
alpha=0.9999951747;
         roc; roccontrast;
run;
```